TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIKKO NIEMINEN

OPC UA -ASIAKASSOVELLUKSEN KEHITTÄMINEN OPC CLASSIC -TEKNIIKOISTA OPC UA:HAN SIIRTYMISEKSI

Diplomityö

Tarkastajat: professori José L. Martinez Lastra ja laboratorioinsinööri Matti Aarnio

Tarkastaja ja aihe hyväksytty
29. marraskuuta 2017

TIIVISTELMÄ

MIKKO NIEMINEN: OPC UA -asiakassovelluksen kehittäminen OPC Classic -tekniikoista OPC UA:han siirtymiseksi
Tampereen teknillinen yliopisto
Diplomityö, 40 sivua
Elokuu 2018
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Factory Automation and Industrial Informatics
Tarkastajat: professori José L. Martinez Lastra ja laboratorioinsinööri Matti Aarnio

Avainsanat: OPC Unified Architecture, OPC Classic, NodeOPCUA, Node.js

Teollisuus 4.0:n eli niin sanotun neljännen teollisen vallankumouksen myötä teollisuuden laitteet ja järjestelmät kytketään verkkoon tietojen välittämistä ja yhteistoimintaa varten. Eri valmistajien laitteiden ja järjestelmien kommunikointiin tarvitaan yhtenäinen ja standardoitu tapa kuvata ja siirtää tietoa. Yksi tällaisista tavoista on OPC (Open Platform Communications). OPC on kokoelma automaation tiedonsiirron spesifikaatioita, jotka määrittävät, miten laitteiden ja järjestelmien tietoja esitetään ja välitetään.

EloWise on modulaarinen tiedonhallintatyökalu, jota käytetään erilaisissa tuotannonohjauksen ja tuotteen elinkaaren hallinnan sovelluksissa. EloWise-tiedonhallintatyökaluun on toteutettu useita yleisiä tiedonsiirtoprotokollia ja -teknologioita käyttäviä moduuleja, jotka mahdollistavat EloWise-tiedonhallintatyökalun yhdistämisen ja yhteistoiminnan eri laitteiden ja järjestelmien kanssa. Aikaisemmin EloWise-tiedonhallintatyökaluun on toteutettu Elomatic Automation Interface -moduuli, joka on mahdollistanut tiedonsiirron OPC Classic -tekniikoihin kuuluvalla OPC Data Access -tekniikalla. Tämän diplomityön tavoitteena oli päivittää EloWise OPC UA -yhdistettävyyttä varten.

Työn ensimmäisessä vaiheessa tehtiin kirjallisuuskatsaus OPC-tekniikoihin, menetelmiin OPC Classic -tekniikoista OPC UA:han siirtymiseen ja saatavilla oleviin OPC UA -kehitystyökaluihin ja -pinoihin. Kirjallisuuskatsauksen pohjalta valittiin EloWise-tiedonhallintatyökalun päivityksessä käytettävä menetelmä OPC Classic -tekniikasta OPC UA:han siirtymiseen. Menetelmäksi valittiin uuden OPC UA -asiakassovellusmoduulin kehitys, jotta aikaisempi toteutus ei rajoittanut kehitystä ja pystyttiin hyödyntämään kaikkia OPC UA:n ominaisuuksia. Menetelmän valinnan jälkeen vertailtiin OPC UA -kehitystyökaluja ja -pinoja asiakassovelluksen kehitystä varten. Kehitykseen valittiin Node.js -pohjainen OPC UA -toteutus, NodeOPCUA, sen riittävien toimintojen ja sallivan Massachusetts Institute of Technology -lisenssin takia.

NodeOPCUA-toteutusta ja muita avoimen lähdekoodin ohjelmakirjastoja käyttäen toteutettiin OPC UA -asiakassovellusmoduuli. Toteutettu OPC UA -asiakassovellusmoduuli mahdollistaa EloWise-tiedonhallintatyökalun yhdistämisen OPC UA -palvelimiin ja sen avulla EloWise-tiedonhallintatyökalun kautta pystytään keräämään tietoja ja hallinnoimaan OPC UA-palvelimien yhteydessä olevia laitteita ja järjestelmiä. Tätä diplomityötä voidaan käyttää apuna OPC UA -asiakassovelluksen kehityksessä tai toteutettaessa OPC Classic -tekniikoista OPC UA:han siirtymistä

ABSTRACT

MIKKO NIEMINEN: Migration from OPC Classic to OPC UA by Developing an OPC UA Client Application
Tampere University of Technology
Master of Science Thesis, 40 pages
August 2018
Master's Degree Programme in Automation Technology
Major: Factory Automation and Industrial Informatics
Examiners: Professor José L. Martinez Lastra and Laboratory Engineer Matti Aarnio

Keywords: OPC Unified Architecture, OPC Classic, NodeOPCUA, Node.js

Industry 4.0 or the so called fourth industrial revolution has propelled industrial devices and systems to be connected to networks for information exchange and co-operation. A common method to represent and exchange information is needed to enable communication between devices and systems made by different manufacturers. One of these ways is OPC (Open Platform Communications). OPC is a collection of specifications for automation which defines how devices and systems represent and exchange information.

EloWise is a modular information management tool used for different tasks in manufacturing execution and product life cycle management. EloWise has multiple modules using common information exchange protocols and technologies to enable connectivity and combability with different devices and systems. Previously a module called Elomatic Automation Interface has been developed to enable information exchange using one of OPC Classic technologies, OPC Data Access. The goal of this thesis is to update the EloWise information management tool for information exchange with OPC UA.

In the first phase of the thesis a literature review was conducted to gather information on OPC technologies, different migration paths from OPC Classic to OPC UA and available OPC UA software development kits and stacks. Based on the literature review a migration method was chosen for the update of the EloWise system. To use all the capabilities of OPC UA and not to be restricted by the previous implementation, the migration was carried out by developing a new OPC UA client application module. Available OPC UA software development kits and stacks were then compared for use in the client application development. The Node.js based implementation NodeOPCUA was chosen because of its adequate features and permissive Massachusetts Institute of Technology licence.

The OPC UA client application module was developed using the NodeOPCUA implementation and other open source libraries. The resulting OPC UA client application module enables the EloWise information management tool to be connected to OPC UA servers and allows management and information exchange with devices and systems connected to these servers. This thesis can be used as a reference when migrating systems from OPC Classic to OPC UA or when developing OPC UA client applications.

ALKUSANAT

Tämä diplomityö on tehty insinööri- ja konsultointitoimisto Elomatic Oy:lle. Haluan kiittää työtovereitani Elomatic Oy:n Turun toimistolta ja erityisesti työtä ohjanneita Markus Havupaloa ja Jussi Soinista neuvoista ja tuesta työn tekemisessä. Lisäksi haluan kiittää työn tarkastajina toimineita professori José L. Martinez Lastraa ja laboratorioinsinööri Matti Aarniota ohjauksesta ja työn tarkastamisesta. Haluan kiittää myös vanhempiani kannustuksesta ja tuesta tämän työn aikana.

Turussa, 22.8.2018

Mikko Nieminen

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

OPC	Open Platform Communications
OPC UA	OPC Unified Architecture
OPC DA	OPC Data Access
OPC A & E	OPC Alarms & Events
OPC HDA	OPC Historical Data Access
COM	Component Object Model
DCOM	Distributed Component Object Model
IEC	International Electrotechnical Commission
EloAI	Elomatic Automation Interface
TCP	Transmission Control Protocol
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
XML	Extensible Markup Language
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
NPM	Node Package Manager
AJAX	Asynchronous JavaScript And XML
SQL	Structured Query Language
MSSQL	Microsoft SQL Server
ORM	Object-relational mapping
GPL	General Public License
LGPL	Lesser General Public License
EPL	Eclipse Public License
RCL	Reciprocal Community License
MIT	Massachusetts Institute of Technology

1. JOHDANTO

Teollisuus 4.0:n eli niin sanotun neljännen teollisen vallankumouksen myötä yhä useampi teollisuuden laite tai järjestelmä kytketään verkkoon tietojen välittämistä ja yhteistoimintaa varten. Jotta eri valmistajien laitteet ja järjestelmät voivat siirtää tietoa ja toimia yhdessä, tarvitaan yhtenäinen ja standardoitu tapa kuvata ja siirtää tietoa. Yksi tällaisista tavoista on OPC (Open Platform Communications). OPC on kokoelma automaation tiedonsiirron spesifikaatioita, jotka määrittävät, miten laitteiden ja järjestelmien tietoja esitetään ja välitetään. OPC-tekniikat perustuvat asiakas-palvelin -malliin. OPC-palvelinsovellus asennetaan laitteen tai järjestelmän yhteyteen ja palvelin tuo niiden tiedot saataville. OPC-asiakassovellus taas ottaa yhteyden OPC-palvelimeen käsitelläkseen siihen yhdistetyn laitteen tai järjestelmän tietoja.

EloWise on Elomaticin modulaarinen tiedonhallintatyökalu erilaisisiin tuotannonohjauksen ja tuotteen elinkaaren hallinnan sovelluksiin. EloWise kerää eri järjestelmien ja laitteiden tietoja yhden hallintatyökalun alle ja mahdollistaa järjestelmien ja laitteiden keskitetyn ohjaamisen ja hallinnoinnin. EloWise- tiedonhallintatyökaluun on aikaisemmin toteutettu EloAI (Elomatic Automation Interface) -moduuli, joka on mahdollistanut tiedonsiirron OPC Classic -tekniikoihin kuuluvalla OPC DA (OPC Data Access) -tekniikalla. OPC DA on tiedonsiirtotekniikkana vanhentunut ja poistumassa käytöstä, joten päivittäminen OPC DA:n seuraajaan, OPC UA:han (OPC Unified Architecture), on tullut ajankohtaiseksi.

Siirtyminen OPC UA:han parantaa EloWise-tiedonhallintatyökalun käytettävyyttä keskitettynä tiedonhallinnan työkaluna usein eri tavoin. OPC UA -liitettävyys mahdollistaa useampien erilaisten laitteiden ja järjestelmien yhteen liittämisen EloWise-tiedonhallintatyökaluun. Lisäksi OPC UA:n avulla voidaan välittää monimuotoisempaa tietoa ja erilaisia tietorakenteita, jotka sisältävät varsinaisen tiedon lisäksi sitä kuvailevaa tietoa.

Tämän diplomityön tavoitteena on EloWise-tiedonhallintatyökalun päivitys OPC UA -liitettävyyttä varten. Työssä selvitetään, miten aikaisemmat OPC Classic -tekniikat poikkeavat OPC UA:sta ja mitä menetelmiä on käytettävissä siirtymiseen OPC Classic -tekniikoista OPC UA:han. Lisäksi selvitetään, mitä kehitystyökaluja on saatavilla OPC UA -sovellusten kehitykseen ja vertaillaan OPC UA -kehitystyökalujen ominaisuuksia. Selvitysten pohjalta valitaan menetelmä OPC DA -tekniikasta OPC UA:han siirtymiseen ja kehitysympäristö sekä -työkalut siirtymisen toteuttamiseen.

Menetelmiä OPC Classic -tekniikoista OPC UA:han siirtymiseen on aikaisemmin esitelty ja vertailtu muutamissa tutkimuksissa [1-3]. Lisäksi Hannelius et. al. toteuttivat tutkimuksessaan [1] esimerkin siirtymisestä adapterisovelluksen avulla. Emdenin ammattikorkeakoulun tutkimuksessa [4] OPC UA -kehitystyökaluja vertailtiin moniosaisella suorituskykytestillä. Lisäksi kehitystyökaluvaihtoehtoja on vertailtu osana tutkimuksia [5-7].

Diplomityö aloitetaan kirjallisuuskatsauksena OPC-tekniikoihin ja menetelmiin OPC Classic -tekniikoista OPC UA:han siirtymiseen sekä saatavilla oleviin OPC UA -kehitystyökaluihin. Kirjallisuuskatsauksessa muodostetaan yleiskuva OPC-tekniikoista ja kerätään tietoja auttamaan tutkimuksen toteuttamisvaiheessa käytettävien menetelmän, kehitysympäristön ja -työkalujen valintaa. Kirjallisuuskatsauksen jälkeen esitellään EloWise-tiedonhallintatyökalu ja EloAI-moduuli ja määritellään vaatimukset toteutukselle. Vaatimusmäärittelyn jälkeen valitaan toteutuksessa käytettävä siirtymismenetelmä OPC Classic -tekniikoista OPC UA:han ja siirtymisessä käytettävät kehitysympäristö ja -työkalut. Lopuksi valitulla siirtymismenetelmällä ja kehitystyökaluilla toteutetaan EloWise-tiedonhallintatyökalun päivittäminen.

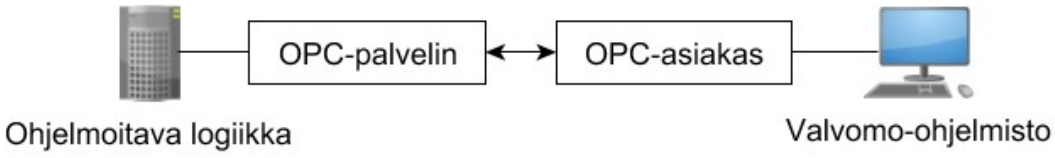
Kirjallisuuskatsauksen tulokset on esitelty luvussa 2. Työn lähtökohtana olevat EloWise-tiedonhallintatyökalu ja siihen aikaisemmin toteutettu EloAI-moduuli, toteutuksen vaatimusten määrittely ja siirtymismenetelmän, kehitysympäristön ja -työkalujen valinta on esitelty luvussa 3. Luvussa 4 esitellään kehitystyön tuloksena syntynyt sovellus ja luvussa 5 on yhteenveto diplomityöstä.

2. LÄHTÖKOHDAT

2.1 OPC-tekniikat

OPC on kokoelma automaation tiedonsiirron spesifikaatioita, joiden tavoitteena on mahdollistaa eri valmistajien laitteiden ja järjestelmien välinen tiedonsiirto ja yhteistoimivuus. OPC on alun perin lyhenne englanninkielisestä termistä OLE (Object Linking and Embedding) for Process Control. Myöhemmin lyhenteen termiä on vaihdettu, ensin termiin Openness, Productivity and Connectivity ja viimeisimpänä termiin Open Platform Communications. Yleisimmin käytetään kuitenkin pelkkää lyhennettä.

OPC-tekniikat noudattavat asiakas-palvelin -mallia eli on olemassa sekä OPC-asiakas- että OPC-palvelinsovelluksia, jotka kommunikoivat keskenään. OPC-palvelinsovellus asennetaan laitteen tai järjestelmän, esimerkiksi ohjelmoitavan logiikan, yhteyteen ja palvelin tarjoaa siihen yhteydessä olevan laitteen tai järjestelmän tiedot asiakassovellusten käsiteltäväksi. OPC-asiakassovellus, joka on esimerkiksi osa valvomo-ohjelmistoa, taas ottaa yhteyden OPC-palvelimeen käsitelläkseen siihen yhteydessä olevan laitteen tai järjestelmän tietoja. Esimerkkitapauksessa OPC-tekniikan avulla valvomo-ohjelmistossa pystytään näyttämään ohjelmoitavan logiikan tietoja. Esimerkkitapaus OPC-tekniikan käytöstä valvomosovelluksessa on havainnollistettu kuvassa 1.



Kuva 1: Esimerkki OPC – tekniikan käytöstä valvomosovelluksessa

OPC-tekniikat voidaan jakaa nykyiseen OPC UA:han ja sitä edeltäneisiin OPC Classic -tekniikoihin. OPC-tekniikoiden kehityksestä ja ylläpidosta vastaa OPC-säätiö. OPC-säätiö, OPC Classic -tekniikat ja OPC UA on esitelty tarkemmin seuraavissa alaluvuissa.

2.1.1 OPC-säätiö

Vuonna 1995 yritykset Fisher-Rosemount, Intellution, Intuitive Technology, Optp22, Rockwell ja Siemens Ag perustivat työryhmän kehittämään standardia reaaliaikaisen automaatiodatan lukemiseen Windows-käyttöjärjestelmillä. Kehityksen tuloksena syntyi ensimmäinen OPC-spesifikaatio, OPC DA, joka julkaistiin elokuussa 1996. Syyskuussa 1996 perustettiin OPC-säätiö vastaamaan OPC-spesifikaatioista ja niiden kehityksestä ja OPC-tekniikoiden markkinoinnista. OPC-säätiö on voittoa tavoittelematon organisaatio,

jonka jäseninä on satoja yrityksiä, oppilaitoksia ja muita organisaatioita. OPC-säätiön tehtävänä on myös OPC-tuotteiden sertifioinnista vastaaminen. OPC-tuotteita voidaan testata säätiön tarjoamilla testityökaluilla tai säätiön hyväksymissä testilaboratorioissa. [8, s.3, 6] [9, s.1-2]

Säätiön sivulla ylläpidetään luetteloa sen jäsenten OPC-tuotteista. Sertifioidut tuotteet on listattu erikseen ja säätiö suosittelee niiden käyttöä sertifioimattomien tuotteiden sijaan. Sertifiointi varmistaa, että tuote noudattaa OPC-spesifikaatioita ja toimii yhdessä muiden OPC-pohjaisten tuotteiden kanssa. Se myös takaa, että tuote toimii vakaasti ja käyttää resursseja tehokkaasti. Sertifioidun tuotteen tunnistaa säätiön myöntämästä sertifiointitunnuksesta. [10, 11]

2.1.2 OPC Classic -tekniikat

Microsoftin COM (Component Object Model) on oliopohjainen standardi, joka määrittää oliomallin ja toteutusvaatimukset binäärisille COM-olioille, jotka voivat kommunikoida keskenään. COM-olio sisältää tietoja ja funktioita, joilla olion tietoihin pääsee käsiksi ja joilla niitä on mahdollista muokata. COM-oliot voivat sijaita samassa suoritettavassa ohjelmassa tai eri ohjelmissa, joita suoritetaan samalla laitteella. DCOM (Distributed Component Object Model) mahdollistaa ohjelmien välisen kommunikoinnin eri laitteiden välillä. COM ja DCOM ovat binäärisiä standardeja eli ne pätevät käännettyyn konekieleen. Tämän ansiosta toistensa kanssa yhteensopivia COM- ja DCOM-oliokirjastoja voidaan toteuttaa eri ohjelmointikielillä. [12]

OPC Classic on yläkäsite, joka kerää yhteen Microsoftin COM- ja DCOM-tekniikoihin perustuvat OPC UA:ta edeltävät tekniikat. OPC Classic -spesifikaatiot rakentuvat perusspesifikaatioiden OPC Overview ja OPC Common päälle. Niistä merkittävimpiä ovat OPC DA, OPC A&E (OPC Alarm & Events) ja OPC HDA (OPC Historical Data Access). OPC DA:ta käytetään prosessidatan välittämiseen, OPC A & E:tä tapahtumatietojen välittämiseen ja OPC HDA:ta arkistoidun tiedon välittämiseen. Lisäksi OPC DA -spesifikaatiota on laajennettu spesifikaatioilla OPC Complex Data, OPC Batch ja OPC Data eXchange. [9]

Kaikissa Windows-pohjaisissa järjestelmissä olevan valmiin ratkaisun hyötykäyttäminen nopeutti OPC-spesifikaatioiden kehittämistä merkittävästi. COM- ja DCOM-olioiden konfigurointi ja ylläpito on kuitenkin monimutkaista ja niiden tietoturvallisuudessa on puutteita. Lisäksi COM/DCOM-pohjaisuus sitoo OPC Classic -tekniikat käytettäväksi pelkästään Windows-käyttöjärjestelmien kanssa. OPC Classic -tekniikoiden avulla ei pystytä esittämään monimutkaisia tietorakenteita ja hierarkioita eikä esittämään varsinaisen tiedon lisäksi siihen liittyvä kuvailevaa tietoa. [1, s.3-4] [9, s.4, 11-15] [13, s.1]

OPC Classic -tekniikoiden COM/DCOM-riippuvuuden ja puutteellisten tiedonmallinnusominaisuuksien takia OPC Classic -tekniikoille kehitettiin seuraaja, OPC UA, joka kokoaa aikaisempien OPC Classic -tekniikoiden ominaisuudet ja toiminnot yhteen spesifikaatioon ja lisää uusia ominaisuuksia ja toimintoja. OPC UA on esitelty seuraavassa alaluvussa.

2.1.3 OPC Unified Architecture

OPC UA on vuonna 2008 julkaistu arkkitehtuuri, joka yhdistää OPC Classic -spesifikaatiot yhdeksi laajennettavaksi sovelluskehykseksi. OPC UA tarjoaa OPC Classic -tekniikoita vastaavat toiminnot ja sisältää myös parannuksia ja uusia toimintoja. Uudet COM/DCOM-tekniikat korvaavat tiedonsiirtotavat tekevät OPC UA:sta alustasta ja käyttöjärjestelmästä riippumattoman ratkaisun, jota voidaan käyttää esimerkiksi perinteisten tietokonejärjestelmien lisäksi sulautetuissa järjestelmissä. [14]

OPC UA:n tärkeimpiä eroavaisuuksia OPC Classic -tekniikoihin ovat COM/DCOM-standardin korvaavat tiedonsiirtotavat. OPC UA -sovellukset voivat välittää tietoja joko binääriseksi tai XML (Extensible Markup Language) -muotoon koodattuna. [9, s. 192-194] Tiedonsiirtoa varten on kaksi menetelmää eri käyttötapauksia varten. UA TCP on TCP-protokollasta (Transmission Control Protocol) muutamia OPC UA:n vaatimuksia varten laajennettu yksinkertainen ja nopea tiedonsiirtoprotokolla. Toinen tiedonsiirtotapa on SOAP/HTTP (Simple Object Access Protocol over Hypertext Transfer Protocol), jota käytetään OPC UA -sovellusten väliseen kommunikointiin internetin välityksellä. SOAP/HTTP soveltuu hyvin käytettäväksi palomuurien kanssa, koska SOAP-viestit välitetään http-protokollaa käyttäen. Näin OPC UA -sovelluksen tiedonsiirto onnistuu yleisten verkkoliikenteen porttien 80 ja 443 kautta. [9, s. 198-200]

OPC UA mahdollistaa sekä yksinkertaisten että monimutkaisten tietojen ja tietorakenteiden esittämisen ja pelkän tiedon lisäksi voidaan tarjota tietoa kuvaavaa tietoa eli niin sanottua metadataa. OPC UA:ssa osoiteavaruus koostuu erilaisista solmuista, jotka liittyvät toisiinsa erityyppisillä viittauksilla ja muodostavat hierarkioita. Samat solmut voivat olla osina erilaisia rakenteita eli sama tieto voi olla saatavilla osoiteavaruudessa eri tavoilla eri käyttötapauksia varten. Tärkeimmät solmutyypit ovat objekti ja muuttuja. Muuttujat pitävät sisällään tietoa ja objektit ovat kokoelmia toisista objekteista ja muuttujista. [9, s.19-20,30] Muuttujille on mahdollista määrittää tietotyyppejä, jotka noudattavat tiettyä rakennetta. Yksittäinen muuttuja on instanssi määritetystä tietotyypistä. OPC UA tarjoaa joukon perustietotyyppejä ja mahdollistaa uusien tietotyyppien luomisen. Määritetyt tietotyypit ovat saatavilla muuttujien lailla OPC UA -osoiteavaruudessa. [9, s.19-20]

OPC Classicin COM-pohjaisen API (Application Programming Interface)-rajapinnan sijaan kommunikaatio OPC UA:ssa tapahtuu palveluiden avulla. OPC UA määrittelee joukon palveluita, joita palvelimet voivat tarjota asiakassovelluksille ja palvelimet ilmoittavat, mitä palveluja ne tukevat. Asiakassovellukset kutsuvat palveluita ja antavat niille

parametrit pyynnöillä ja saavat palvelun tuloksen vastauksena. Palvelut on rajattu pieneen joukkoon yleiskäyttöisiä metodeja, joita käytetään useisiin käyttötarkoituksiin sen sijaan, että jokaiselle toiminnolle olisi oma metodinsa. [9, s.125]

OPC UA on jaettu osa-alueittain spesifikaatioihin, jotka tunnetaan myös IEC (International Electrotechnical Commission) 62541-alkuisina standardeina. Spesifikaatioita on tällä hetkellä yhteensä 14 kappaletta, joiden lisäksi on tarjolla eri standardeja ja tekniikoita, kuten ISA-95 ja IEC 61850, varten toteutettuja lisäspesifikaatioita. Taulukossa 1 on listattu nämä 14 pääspesifikaatiota ja lyhyt kuvaus niiden sisällöstä. [9, s.11-12] [15]

Taulukko 1: IEC62541 Standardit / OPC UA – spesifikaatiot, perustuu lähteeseen [15]

Spesifikaatio	**Kuvaus**
Part 1: Overview and Concepts	Yleistason esittely OPC UA -tekniikasta
Part 2: Security Model	Turvallisuuden vaatimukset ja malli OPC UA:ssa
Part 3: Address Space Model	Osoiteavaruuden ja tyyppitietojen kuvaus
Part 4: Services	OPC UA -palveluiden kuvaukset
Part 5: Information Model	Tiedon mallintaminen OPC UA:ssa
Part 6: Mappings	Tiedonsiirto OPC UA -palvelimen ja -asiakkaan välillä
Part 7: Profiles	Profiilien eli OPC UA -toimintokokonaisuuksien määrittelyt
Part 8: Data Access	Automaatiodatan käsittely
Part 9: Alarms and Conditions	Hälytys- ja tapahtumadatan käsittely
Part 10: Programs	Ohjelmat
Part 11: Historical Access	Arkistoidun tiedon käsittely
Part 12: Discovery and Global Services	OPC UA -sovellusten etsintä ja hallinta
Part 13: Aggregates	Koostefunktiot
Part 14: PubSub	PubSub-kommunikaatio

2.2 OPC UA -sovelluksen rakenne

OPC UA -sovellus koostuu kolmesta ohjelmakerroksesta. Ensimmäisenä kerroksena on OPC UA -pino (engl. stack), joka vastaa alemman tason funktioista, kuten viestien koodauksesta, salauksesta ja lähettämisestä. Seuraavana tasona ovat pinoa hyödyntävät OPC UA -kehitystyökalut (engl. software development kit), jotka tarjoavat ylemmän tason funktioita OPC UA -palveluihin ja toimintoihin, kuten yhteyksien hallinnointiin ja viestien käsittelyyn. Pinon ja kehitystyökalujen päälle rakennetaan varsinainen käyttötapauskohtainen OPC UA -asiakas- tai OPC UA -palvelinsovellus, joka välittää ja käyttää tietoa OPC UA:n avulla. [9, s.14,255] OPC UA -sovelluksen ohjelmakerrokset on havainnollistettu kuvassa 2.

Kuva 2: OPC UA -sovellusten ohjelmakerrokset, perustuu lähteeseen [9, s.14]

Samaa pinoa voidaan käyttää sekä palvelin- että asiakassovelluksessa, mutta osa toiminnoista on pelkästään asiakas- tai palvelinsovelluksia varten. Tämän takia pinossa on oma rajapintansa kumpaakin varten. Muu pino voidaan jakaa omiin kerroksiinsa, jotka vastaavat viestien koodauksesta ja dekoodauksesta, niiden turvallisuudesta ja niiden lähettämisestä ja vastaanottamisesta. Nämä kerrokset toteutetaan toimimaan alustasta riippumatta ja lisäksi toteutetaan alustakohtainen kerros. Näin samaa pinoa pystytään käyttämään eri alustoilla pelkästään alustakohtaista kerrosta muuttamalla. [9, s.256-258]

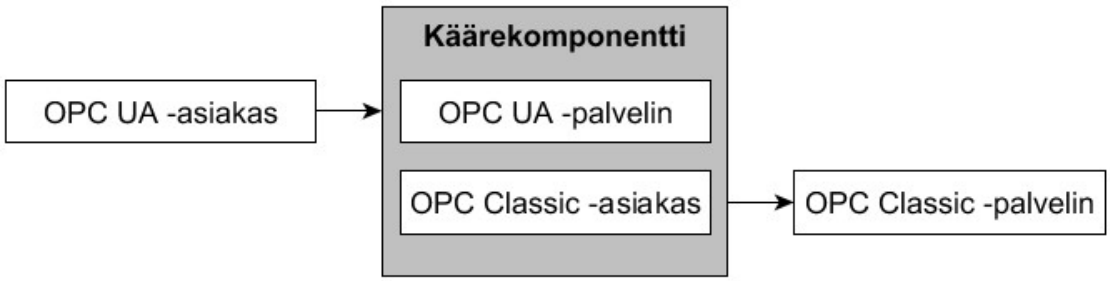
2.3 Siirtyminen OPC Classic-tekniikoista OPC UA:han

OPC Classic -tekniikat ja erityisesti OPC DA ovat olleet laaja-alaisesti käytössä. Olemassa olevien OPC-sovellusten päivittämiseen käyttämään OPC UA:ta on useita, erilaisiin tilanteisiin soveltuvia vaihtoehtoja. Yksi keino on käyttää OPC-säätiön tai muun julkaisijan tekemiä kääre- ja välityskomponentteja olemassa olevien ohjelmien nopeaan integrointiin OPC UA:ta käyttäviin järjestelmiin. Toinen vaihtoehto on toteuttaa kääre- ja välityskomponenttien pohjana olevat muunnokset OPC Classicista OPC UA:han ole-

massa olevaan ohjelmaan, jolloin ei tarvitse käyttää lisäkomponentteja varsinaisen ohjelman rinnalla. Viimeinen vaihtoehto on toteuttaa siirtyminen OPC Classicista OPC UA:han kehittämällä OPC UA -toiminnot alusta alkaen osaksi olemassa olevaa sovellusta tai kehittää kokonaan uusi OPC UA -sovellus. [9, s.283, 293] Eri siirtymisvaihtoehtoja on esitetty tarkemmin seuraavissa alaluvuissa.

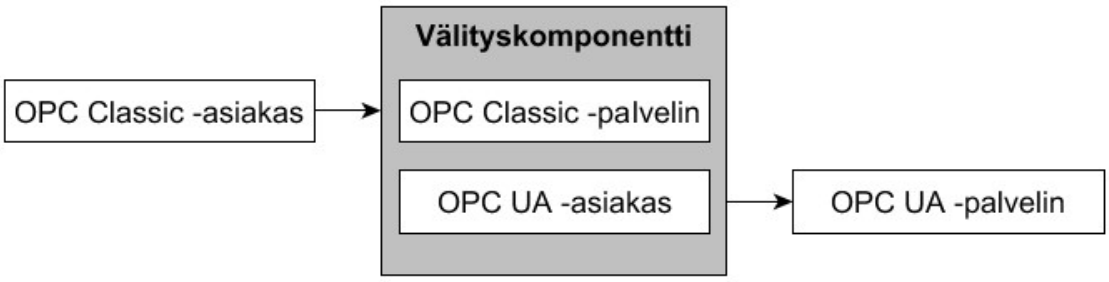
2.3.1 Kääre- ja välityskomponentit

Kääre- ja välityskomponentit ovat varsinaisen OPC-sovelluksen yhteydessä käytettäviä lisäohjelmia, jotka mahdollistavat OPC Classic - ja OPC UA -sovellusten välisen kommunikoinnin ilman muutoksia itse sovellukseen. Käärekomponentti (engl. wrapper) on OPC UA -palvelinsovellus, johon on yhdistetty OPC Classic -asiakassovelluksen toiminnot. OPC UA -asiakassovellus pystyy käärekomponentin avulla yhdistämään OPC Classic -palvelimelle. Käärekomponentti hoitaa OPC UA -pyyntöjen muuntamisen OPC Classic -pyynnöiksi ja muuttaa vastaukset OPC UA -vastauksiksi. Käärekomponentti vastaa myös COM-palvelimen osoiteavaruuden muuntamisesta OPC UA:han. [1, s. 3] [3, s.2] Käärekomponentin toimintaa on havainnollistettu kuvassa 3.



Kuva 3: Käärekomponentin toiminta, perustuu lähteeseen [3, s.2]

Vastaavasti OPC Classic -asiakassovellus voi yhdistää OPC UA -palvelimelle välityskomponentin (engl. proxy) avulla. Välityskomponentti on OPC Classic -palvelinsovellus, johon on yhdistetty OPC UA -asiakassovelluksen toiminnot. Se hoitaa OPC Classic -pyyntöjen muuntamisen OPC UA -pyynnöiksi ja vastausten muuntamisen OPC Classic -palautusarvoiksi. Lisäksi välityskomponentti muuntaa OPC UA -palvelimen osoiteavaruuden COM-standardin mukaiseksi [1, s. 3] [3, s.2] Välityskomponentin toimintaa on havainnollistettu kuvassa 4.



Kuva 4: Välityskomponentin toiminta, perustuu lähteeseen [3, s.2]

OPC UA -palvelinta voidaan käyttää myös yhdyskäytävänä COM-pohjaisiin palvelinsovelluksiin. Yhdyskäytävässä useampi käärekomponentti integroidaan saman OPC UA -palvelinsovelluksen yhteyteen. Tällainen yhdyskäytävä mahdollistaa OPC UA -asiakassovellukselle yhtenäisen pääsyn OPC DA -, OPC A & E - ja OPC HDA -palvelimille. Erityistapaus yhdyskäytävästä on räätälöity sovitinsovellus. Sovitinsovelluksessa muunnosten sijaan toteutetaan palveluita OPC Classic -sovellusten kanssa suoraan kommunikointiin. Esimerkki tällaisesta sovitinsovelluksesta toteutettiin osana tutkimusta [1, s 3.-5].

Kääre- ja välityskomponenttien etu on niiden käytön nopeus ja helppous. Olemassa olevia OPC Classic -tekniikoihin perustuvia sovelluksia pystytään niiden avulla käyttämään ilman muutoksia sovelluksen toimintaan vain lisäämällä tarvittava kääre- tai välityskomponentti sovelluksen rinnalle OPC UA -sovelluksiin yhdistämistä varten. Kääre- ja välityskomponenttien avulla ei kuitenkaan pystytä käyttämään kaikkia OPC UA:n tarjoamia uusia ominaisuuksia ja toimintoja. Lisäksi yhtenäinen eri OPC Classic -tekniikoiden käyttö samasta sovelluksesta ei ole mahdollista vaan OPC Classic -tekniikoilla pääsy reaaliaikaisiin tietoihin, hälytyksiin ja tapahtumiin ja historialliseen tietoon täytyy toteuttaa omien palvelimien kautta, jotka kaikki noudattavat omia spesifikaatioitaan. Näiden ominaisuuksien samanaikainen toteuttaminen kääre- ja välityskomponenttien avulla vaatii paljon konfigurointia ja tekee sovelluksen ylläpidosta monimutkaisempaa. Kääre- ja välityskomponentit lisäävät myös liikennettä, joka vaaditaan samojen viestien lähettämiseen, sillä jokainen viesti joudutaan muuntamaan tekniikasta toiseen, jonka jälkeen täytyy lähettää uusi viesti eteenpäin. Näistä syistä niiden käyttöä suositellaan lähinnä vanhojen järjestelmien tuen jatkamiseen. [1, s.296] [3, s.4] [9, s.2]

Kääre- ja välityskomponenttien pohjana olevat muunnokset OPC Classic -tekniikoista OPC UA:han voidaan toteuttaa myös suoraan osaksi olemassa olevaa sovellusta. Tällöin sovelluksen yhteyteen lisättävä kääre- tai välityskomponentti ei rajoita sovelluksen toimintaa ja suorituskykyä ja tee sen ylläpidosta hankalampaa. [9, s.293]

2.3.2 OPC UA -kehitystyökalut ja -pinot

Siirtyminen voidaan toteuttaa myös toteuttamalla OPC UA -toiminnot aikaisemmin toteutettujen OPC Classic -toimintojen rinnalle olemassa olevaan sovellukseen, korvaamalla vanhat toiminnot OPC UA -toiminnoilla tai kehittämällä kokonaan uusi vanhan sovelluksen korvaava OPC UA:ta käyttävä sovellus. Kehitystyö vie enemmän aikaa kuin kääre- ja välityskomponenttien tai muunnosten avulla siirtymisen toteuttaminen, mutta tällöin pystytään hyödyntämään kaikkia OPC UA:n tarjoamia ominaisuuksia ja toimintoja. OPC UA:n yleiskäyttöinen ja laajennettava rakenne mahdollistaa kehitystyön jakamisen pienempiin kokonaisuuksiin ja ominaisuuksien lisäämisen yksi kerrallaan pidemmän ajan kuluessa. Kehitystyön alussa voidaan korvata OPC Classic -toiminnot vastaavilla OPC UA -toiminnoilla ja lisätä OPC UA:n tarjoamia uusia toimintoja vasta jatkokehityksen yhteydessä. Esimerkiksi OPC DA:n toiminnoille kuten ryhmien ja alkioiden lisäämiselle tietomuutosten seuraamista varten löytyy suoraan vastine OPC UA:sta. [9, s.296,319-320]

Olemassa olevien OPC UA -kehitystyökalujen ja -pinojen käyttö helpottaa kehitystyötä ja toimintojen integrointia. OPC UA -kehitystyökaluja ja -pinoja kehittävät OPC-säätiö sekä useat yritykset, yhteisöt ja yliopistot. Saatavilla oleviin OPC UA -kehitystyökaluihin ja -pinoihin tutustuttiin ja niitä vertailtiin tutkimusten, OPC-säätiön tuoteluettelon ja verkkosivuston ja kehitystyökalujen verkkosivustojen ja dokumentaatioiden avulla.

OPC-säätiö tarjoaa ANSI-C:llä, C#:lla ja Javalla toteutettuja pinoja. Pinoja varten ei kuitenkaan ole kehitystyökaluja, mutta niiden käytöstä on tarjolla esimerkkisovelluksia ja -ohjelmakoodeja. OPC-säätiön tarjoamat pinot ovat käytettävissä kaksoislisenssillä. Ilman jäsenyyttä ne ovat käytössä GPL (General Public License) -lisenssillä, eli pinoja käyttävä ohjelmakoodi tulee julkaista avoimena. OPC-säätiön yritysjäsenille lisenssi on RCL (Reciprocal Community License) eli pinoja voi käyttää ilman pinoja käyttävän ohjelmakoodin jakamista. Kuitenkin mahdolliset muutokset pinoihin täytyy tuoda julkiseksi. Jos jäsenyys loppuu, pinojen versiota ei voi päivittää ilman lisenssin vaihtumista GPL -lisenssiin. [16]

Kaupallisia OPC UA -kehitystyökaluja tarjoavat useat yritykset. OPC-säätiön ylläpitämän tuoteluettelon [11] kautta löytyviä kaupallisia kehitystyökaluvaihtoehtoja on kerätty taulukkoon 2.

Taulukko 2: Kaupalliset OPC UA -toteutukset, perustuu lähteeseen [11]

Valmistaja	**Ohjelmointikielet**
Advosol	.NET
Matrikon	C++
OPCLabs	.NET
Prosys OPC	Java
Softing	C++, .NET
Technosoftware	.NET
Unified Automation	Ansi C, C++, .NET, Java

Lisäksi saatavilla on avoimen lähdekoodin OPC UA -pinoja ja -kehitystyökaluja useille eri ohjelmointikielille. OPC UA -kehitystyökaluja ja -pinoja on vertailtu muutamissa aikaisemmissa tutkimuksissa. Palm et. al. esittelevät ja vertailevat tutkimuksessaan [5] vuonna 2015 saatavilla olleita avoimen lähdekoodin OPC UA -toteutuksia. Vertailun pohjalta huomattiin tarve avoimen lähdekoodin OPC UA -toteutukselle, jota voidaan käyttää millä tahansa prosessorilla ja joka vaatii vain vähän muistia ja laskentatehoa käytettävältä järjestelmältä. Perustettiin open62541-projekti, jonka tuloksena kehitetty uusi open62541-kehitystyökalu C-ohjelmointikielelle esitellään osana tutkimusta. Open62541-kehitystyökalu on jatkuvassa kehityksessä ja sen GitHub-sivulla [17] ylläpidetään listaa muista saatavilla olevista avoimen lähdekoodin OPC UA -toteutuksista ja niiden lisensseistä.

Fortiss GmbH:n tutkimuksessa [6] vertaillaan avoimen lähdekoodin C/C++- ja Java-pohjaisia OPC UA -kehitystyökaluja keskittyen lähinnä palvelinten löytämisen palveluihin. Tutkimuksen toteutusvaiheessa päädyttiin käyttämään open62541-kehitystyökalua ja Java-pohjaista Eclipse Milo -kehitystyökalua niiden sallivien lisenssiehtojen ja riittävien toimintojen takia. Schulz et. al. vertailevat lyhyesti OPC UA -kehitystyökaluja osana OPC UA -sovellusten kehittämistä käsittelevää tutkimustaan [7].

Muihin tutkimuksiin verrattuna kattavammin OPC UA -kehitystyökaluja ja -pinoja vertailtiin Emdenin ammattikorkeakoulun tutkimuksessa [4] . Tutkimuksessa tehtiin 21 eri OPC UA -toteutukselle kattava suorituskykytesti. OPC UA -toteutuksia vertailtiin 15 eri kriteerillä, jotka olivat:

1. toteutuksen versio
2. viimeisimmän päivityksen päivämäärä
3. onko toteutuksen kehittäjänä yksityishenkilö, yhteisö, tutkimuslaitos vai yritys
4. ohjelmointikieli
5. lisenssi avoimen lähdekoodin toteutuksille
6. kokeiluversion saatavuus kaupallisille toteutuksille
7. onko toteutus sertifioitu
8. onko toteutus pelkästään pino vai kehitystyökalut ja onko kehitystyökalut saatavilla palvelin- ja/tai asiakassovelluksia varten
9. toteutetut protokollavaihtoehdot
10. toteutetut palvelut joukosta peruspalveluja
11. turvallisuusvaihtoehdot
12. käyttäjien tunnistus
13. dokumentaation taso
14. tuetut käyttöjärjestelmät
15. TRL (Technology Readiness Level)

Suorituskykytesti tehtiin katsauksena toteutuksista saatavilla oleviin dokumentaatioihin, verkkosivustoihin ja lähdekoodeihin. Testissä ei otettu huomioon keskeneräisiä ja kehityksessä olevia toimintoja. TRL:n määritys tehtiin toteutuksien kehittäjiä haastattelemalla.

Vain kolme toteutuksista, Unified Automation:in C++-pohjainen kehitystyökalu, Java-pohjainen Eclipse Milo-kehitystyökalu ja OPC-säätiön .NET -pino, täytti kaikki suorituskykytestin kriteerit. Tosin kriteereistä osa oli ongelmallisia mitata ja vertailla. Esimerkiksi dokumentaatioiden laajuus on vaikea määrittää ja niitä on vaikea vertailla, koska jokainen kehittäjä on kuvannut toteutuksensa omalla tavallaan. Harvalle toteutuksista saatiin määritettyä TRL vertailua varten.

Vertailussa muutama avoimen lähdekoodin toteutus, OPCUA4j ja OPyCua, erottui selvästi muista. Niitä ei ole päivitetty pitkään aikaan ja niistä puuttuu useita tärkeitä ominaisuuksia ja toimintoja. Muuten avoimen lähdekoodin toteutuksiin on toteutettuna useimmat tarkastelluista palveluista ja osa muista toiminnoista. Kuitenkin useissa avoimen lähdekoodin toteutuksista on saatavilla vain rajallinen dokumentaatio.

Kaupallisten toteutusten välillä ei ollut suuria eroja. Eroavaisuudet toteutusten välillä ovat lähinnä palvelujoukkojen sijaan muissa tarkastelluissa toiminnoissa. Kaupallisista toteutuksista kaikki yhtä lukuun ottamatta ovat OPC-säätiön sertifioimia.

Taulukkoon 3 on koottu lähteiden [4-6, 17] pohjalta listaus eri avoimen lähdekoodin toteutusvaihtoehdoista ja niiden ohjelmointikielistä ja lisensseistä.

Taulukko 3: Avoimen lähdekoodin OPC UA -toteutukset, perustuu lähteisiin [4-6, 17]

	Ohjelmointikielet	**Lisenssi**
ASNeG	C++	Apache
Eclipse Milo	Java	EPL
FreeOpcUa	C++, Python	LGPL
NodeOPCUA	JavaScript	MIT
OPC Foundation	.NET, Java, C	GPL/ RPC
opcua4j	Java	CC 3.0 BY-SA
OPC UA Client	C#	MIT
open62541	C	MPL-2.0
OpenOpcUa	C++	Cecil-C, kertamaksu
OpenScada UA Interface	C++	GPL
UACL	C#, C++	LGPL
UAF	C++/Python	LGPL

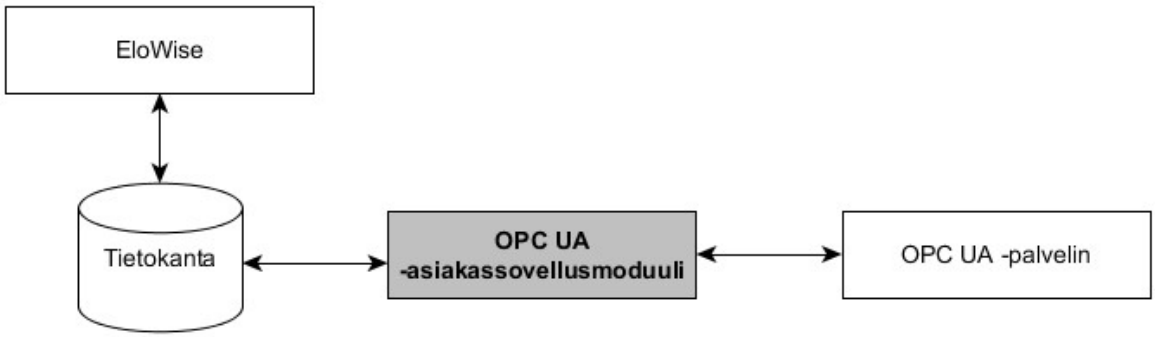
3. KÄYTETYT MENETELMÄT JA KEHITYSTYÖKALUT

3.1 EloWise- tiedonhallintatyökalu

EloWise on modulaarinen tiedonhallintatyökalu, jota käytetään tuotannonohjauksen ja tuotteen elinkaaren hallinnan sovelluksissa. EloWise kokoaa useiden laitteiden ja järjestelmien tietoja yhden hallintatyökalun alle ja mahdollistaa laitteiden ja järjestelmien keskitetyn ohjaamisen ja hallinnoinnin samalla työkalulla. EloWise- tiedonhallintatyökaluun on toteutettu useita yleisiä tiedonsiirtoprotokollia ja -teknologioita hyödyntäviä ohjelmamoduuleja laitteisiin ja järjestelmiin yhdistämisen mahdollistamiseksi.

EloWise- tiedonhallintatyökaluun on aikaisemmin toteutettu OPC DA -liitettävyyttä varten EloAI -moduuli. EloAI-moduuli on OPC DA -asiakassovellus, joka on modulaarinen osa EloWise-tiedonhallintatyökalua, mutta sitä voidaan käyttää myös erillisenä, itsenäisenä ohjelmana. EloAI-moduuli mahdollistaa tietojen lukemisen ja kirjoittamisen ja aika- ja liipaisupohjaisen tietojen tilaamisen OPC DA -palvelimilta. Lisäksi sen avulla luettuja tietoja ja tiedonkeruun konfigurointeja voidaan kerätä Microsoft SQL Server -tietokantaan.

Tämän diplomityön tavoitteena on toteuttaa aikaisemmin toteutetun OPC DA -tekniikkaa käyttävän EloAI-moduulin päivittäminen OPC UA -asiakassovellusmoduuliksi. OPC UA -asiakassovellusmoduuli mahdollistaa EloWise-tiedonhallintatyökalun yhdistämisen OPC UA -palvelimiin ja niiden yhteydessä olevien laitteiden ja järjestelmien hallinnoinnin ja tietojen käsittelyn. Päivitettyä moduulia käytetään myös vanhan moduulin lailla sekä osana EloWise- tiedonhallintatyökalua että itsenäisenä OPC UA -asiakassovelluksena. OPC UA -asiakassovellusmoduulin rooli EloWise-tiedonhallintatyökalussa on havainnollistettu kuvassa 5.



Kuva 5: Toteutettavan sovelluksen rooli EloWise- tiedonhallintatyökalussa

3.2 Menetelmän valinta OPC UA:han siirtymiseen

Luvussa 2.3 esiteltiin ja vertailtiin eri siirtymisvaihtoehtoja OPC Classic -tekniikoista OPC UA:han ja saatavilla olevia OPC -kehitystyökaluja ja -pinoja. Vertailun pohjalta tuli päättää, mitä vaihtoehtoa EloWise- tiedonhallintatyökalun päivityksen toteutuksessa käytetään siirtymiseen OPC UA:han.

Kääre- ja välityskomponentit ovat siirtymismenetelmistä nopein ja helpoin vaihtoehto. Niiden avulla siirtyminen pystytään toteuttamaan vain lisäämällä tarvittava komponentti olemassa olevan sovelluksen yhteyteen. Ne eivät kuitenkaan ole laajennettavuuden ja ylläpidon kannalta toimiva ratkaisu. Kääre- ja välityskomponentit lisäävät ylimääräisen ylläpidettävän osan järjestelmään eikä niitä käyttämällä kaikkien OPC UA -toiminnallisuuksien toteuttaminen ole mahdollista.

Yksi vaihtoehto on muokata olemassa oleva EloAI-moduuli käyttämään OPC UA:ta. Muokkaamalla EloAI-moduulia olemassa olevaa toteutusta pystyttäisiin hyödyntämään soveltuvilta osin. EloAI-moduulin muokkauksessa tosin nykyinen toteutus rajaa kehityksen ja kehitystyökaluvaihtoehdot .NET-kehitysympäristöön ja vaatii aikaa kehitystyön lisäksi EloAI-moduulin olemassa olevien ohjelmakoodien läpikäymiseen ja muilta osin päivittämiseen.

Siirtymismenetelmäksi valittiin kokonaan uuden OPC UA -asiakassovellusmoduulin kehittäminen. Uuden moduulin kehittämiseen vaadittava aika on pidempi verrattuna muihin vaihtoehtoihin. Uuden, erillisen moduulin toteuttaminen mahdollistaa kuitenkin kehityksen ilman vanhan toteutuksen rajoitteita ja kehitysympäristö ja -työkalut voidaan valita kaikkien saatavilla olevien vaihtoehtojen joukosta. Tarpeen vaatiessa olemassa olevaa moduulia voidaan käyttää uuden moduulin kanssa samanaikaisesti. Uudessa moduulissa pystytään hyödyntämään kaikkia OPC UA:n ominaisuuksia ja toiminnallisuuksia ja kehitykseen vaadittavaa aikaa voidaan vähentää rajaamalla aluksi toteutettavan moduulin vaatimuksia ja lisäämällä uusia ominaisuuksia tarpeen vaatiessa myöhemmin

3.3 Vaatimukset OPC UA - asiakassovellusmoduulille

Siirtymismenetelmän valinnan jälkeen määritettiin vaatimukset toteutettavalle OPC UA -asiakassovellusmoduulille. Kehitystyöhön vaadittavan ajan rajaamiseksi toimeksiantajan kanssa päätettiin, että aluksi toteutetaan vain EloAI-moduulia vastaavat toiminnot OPC UA:ta käyttäen ja sovellukselle tehdään pelkästään yksinkertainen käyttöliittymä. Uusia OPC UA:n tarjoamia toimintoja lisätään ja käyttöliittymää päivitetään vasta jatkokehityksen yhteydessä.

EloAI-moduulin toimintojen pohjalta valittiin OPC UA Services -spesifikaation [15] palveluista uuteen moduuliin tarvittavat palvelut. Taulukkoon 4 on listattu valitut palvelujoukot ja niiden palvelut, jotka uuteen moduuliin tulee toteuttaa.

Taulukko 4: Moduuliin vaadittavat palvelut ja palvelujoukot

Palvelujoukko	**Palvelu**
Secure Channel Service Set	OpenSecureChannel, CloseSecureChannel
Session Service Set	CreateSession, ActivateSession, CloseSession, Cancel
View Service Set	Browse, BrowseNext, TranslateBrowsePaths-ToNodeIds
Attribute Service Set	Read, Write
MonitoredItem Service Set	CreateMonitoredItems, ModifyMonitoredItems, SetMonitoringMode, SetTriggering, DeleteMonitored-Items
Subscription Service Set	CreateSubscription,ModifySubscription, SetPub-lishingMode, DeleteSubscriptions

EloWise ja nykyinen EloAI-moduuli käyttävät MSSQL (Microsoft SQL Server) -tietokantaa tietojen ja konfiguraatioiden säilömiseen. Jotta tiedonkeruu ja tiedonsiirto uuden OPC UA -asiakassovellusmoduulin ja muiden EloWise-moduulien välillä olisi mahdollista, moduuliin toteutetaan tietokantaliityntä MSSQL-tietokantaan.

3.4 OPC UA -kehitystyökalun ja -pinon valinta

Uuden OPC UA -asiakasmoduulin toteutusta varten täytyi valita kehityksessä käytettävä OPC UA -kehitystyökalu ja -pino. Kehitystyökaluvalinnan pohjana käytettiin luvun 2.3.2 selvitystä saatavilla olevista OPC UA -kehitystyökaluista ja pinoista. Moduulille edellisessä luvussa asetetut vaatimukset eivät juurikaan rajanneet kehitystyökaluvaihtoehtoja, koska suurimmassa osassa tarjolla olevista kehitystyökaluista on toteutettuna moduuliin tarvittavat palvelut. Lisäksi uusi moduuli on oma erillinen kokonaisuutensa, joten se voidaan käytännössä toteuttaa millä tahansa ohjelmointikielivaihtoehdoista. Kuitenkin aikaisempi kokemus kehitystyökalujen ohjelmointikielistä ja kehitysympäristöistä oli yksi valintakriteereistä, koska uuden ohjelmointikielen ja kehitysympäristön opettelu kehityksen ohessa lisää kehitykseen vaadittavaa aikaa.

Ensimmäinen vaihtoehto olisi ollut käyttää OPC-säätiön tarjoamia pinoja ja ottaa mallia kehitykseen saatavilla olevista esimerkkiohjelmista. OPC-säätiön tarjoamien pinon mukana ei kuitenkaan tule varsinaisia kehitystyökaluja, joten niiden käyttäminen on muihin vaihtoehtoihin verrattuna työläämpää. Lisäksi niiden käyttäminen kaupallisessa sovelluksessa vaatii käytännössä OPC-säätiön yritysjäseneksi liittymistä, koska ilman jäsenyyttä

pinoa käyttävä ohjelmakoodi täytyy jakaa avoimena. OPC-säätiön jäsenyyttä on myös ylläpidon ja päivittämisen kannalta välttämätöntä jatkaa ohjelman elinkaaren ajan, koska pinon päivittäminen vaatii aktiivisen jäsenyyden. Jäsenyyden ylläpidosta koituu vuosittaisia kustannuksia.

Toinen vaihtoehto olisi ollut valita kaupallinen OPC UA -kehitystyökalu ja -pino. Kaupallisiin kehitystyökaluihin on toteutettu kaikki tärkeimmät palvelut. Niiden käyttöön saa tukea helposti kehitystyökalun toteuttaneelta yritykseltä ja niistä on saatavilla kattava dokumentaatio. Kaupallisten kehitystyökalujen ostaminen nostaa kuitenkin moduulin kehityskustannuksia. Sovelluksen kehitykseen ja ylläpitoon vaaditaan kehitystyökalun lisenssi jokaista kehittäjää varten. Lisäksi ohjelma tulee niiden käytön myötä riippuvaiseksi toisen yrityksen tuotteesta. Kaupallisia kehitystyökaluvaihtoehtoja ei myöskään ole saatavilla kuin rajatulle ohjelmointikielivalikoimalle.

Vaihtoehtona oli myös avoimen lähdekoodin OPC UA -kehitystyökalujen käyttäminen. Avoimen lähdekoodin kehitystyökalut ovat ilmaisia ja niitä on saatavilla kaupallisiin kehitystyökaluvaihtoehtoihin verrattuna useammille ohjelmointikielille. Niiden ominaisuudet ja dokumentaatio ovat kuitenkin kaupallisiin kehitystyökaluihin verrattuna rajallisemmat ja niiden päivitysten tiheys on vaihtelevaa. Niiden käyttöön ei myöskään saa tukea yhtä varmasti tai nopeasti.

Osa avoimen lähdekoodin kehitystyökalujen lisensseistä rajoittaa niiden käyttöä kaupallisissa sovelluksissa. Sallivan lisenssin, kuten MIT (Massachusetts Institute of Technology), kehitystyökalujen käyttö ei aseta juurikaan rajoituksia niitä käyttävälle sovellukselle. Riittää, että sovelluksen mukana toimitetaan kopio kehitystyökalukirjaston lisenssiehto- ja tekijänoikeustiedotteesta. Niin sanotut copyleft-lisenssit ovat kaupallisen tuotteen kehityksessä ongelmallisia. Vahvan copyleft-lisenssin, kuten GPL, omaavat kehitystyökalut eivät sovellu ollenkaan kaupallisen tuotteen kehittämiseen, koska kehitystyökalun lisenssin myötä kaikki sitä käyttävä lähdekoodi on julkaistava avoimena. Heikon copyleft-lisenssin, kuten LGPL (Lesser General Public License) ja EPL (Eclipse Public Licence), kehitystyökalujen käyttö kaupallisen tuotteen kehityksessä on mahdollista, mutta jos kehitystyökalukirjasto jaetaan sovelluksen mukana, täytyy kehitystyökalukirjaston lähdekoodi ja mahdolliset muutokset siihen sisällyttää avoimena sovelluksen mukaan. [18] Lisäksi esimerkiksi LGPL-lisenssi rajoittaa ohjelman toteutustapaa ja julkaisua, sillä jos kehitystyökalukirjastoon viitataan staattisesti eli kirjasto on kiinteänä osana sovellusta, täytyy sovellus toimittaa sellaisessa muodossa, että käyttäjä pystyy muokkaamaan kehitystyökalukirjastoa ja linkittämään sen uudelleen sovellukseen. [19]

Tutkimuksen toimeksiantajan kanssa päädyttiin käyttämään OPC UA -asiakassovellusmoduulin kehitykseen avoimen lähdekoodin OPC UA -toteutusta, koska sen käytöstä ei aiheudu ylimääräisiä kehityskustannuksia ja moduulin vaatimuksien täyttämiseen riittävät rajallisemmatkin kehitystyökalun ominaisuudet. Luvun 2.3.2 taulukkoon 3 listattuja avoimen lähdekoodin OPC UA -toteutuksia vertailtiin kirjallisuuskatsauksen pohjalta ja

osa toteutuksista rajattiin pois vaihtoehdoista. Tutkimuksen [4] vertailutestin perusteella oli muutama avoimen lähdekoodin toteutus, OPCUA4j ja OPyCua, jotka karsiutuivat pois vaihtoehdoista, koska niitä ei enää päivitetä aktiivisesti ja niiden toteutus on jäänyt kesken. Vaihtoehdoista karsittiin lisäksi toteutukset ASNeG, OPC UA Client ja UAF, joiden dokumentaatio oli suorituskykytestin perusteella puutteellinen.

Kaikissa jäljelle jääneissä vaihtoehdoissa oli toteutettavan OPC UA -asiakassovellusmoduulin kannalta tarvittavat ominaisuudet. Vaihtoehdot poikkesivat toisistaan lähinnä käytetyssä ohjelmointikielessä ja lisenssiehdoissa. Lopulta kehitystyökaluksi valittiin NodeOPCUA sen kaupallisen tuotteen kehitykseen soveltuvan sallivan MIT-lisenssin takia. Valintaa puolsi aikaisempi kokemus kehitystyöstä Node.js-kehitysympäristössä ja JavaScript-ohjelmointikielestä. Kehitystyökalun valintaan vaikutti myös NodeOPCUA-kehitystyökalun aktiiviset päivitykset ja kattava dokumentaatio. Lisäksi NodeOPCUA-kehitystyökalun eduksi katsottiin Node.js:lle tarjolla oleva laaja valikoima muita avoimen lähdekoodin ohjelmakirjastoja, joita voidaan käyttää helpottamaan ja nopeuttamaan moduulin kehitystä. Node.js-kehitysympäristö ja NodeOPCUA-kehitystyökalu on esitelty tarkemmin seuraavissa alaluvuissa.

3.5 Node.js-sovellusalusta

Node.js on Chromen V8-moottoriin pohjautuva, laajennettava JavaScript-sovellusalusta, joka mahdollistaa JavaScriptin suorittamisen selaimen ulkopuolella. Node.js-sovelluksessa ohjelman suoritus on asynkronista. Tavallisemmassa synkronisessa ohjelman suorituksessa ohjelman suoritus etenee funktio kerrallaan ja seuraavaan funktioon siirrytään vasta, kun edeltävä funktio valmistuu. Asynkronisessa ohjelman suorituksessa funktioiden suoritus ei estä muun ohjelman suorituksen jatkumista. Funktion suoritus etenee muun ohjelman jatkuessa ja kun funktion suoritus valmistuu, kutsutaan funktion parametriksi annettua funktiota, jota sanotaan takaisinkutsufunktioksi.

Prosessi on järjestelmässä suorituksessa oleva ohjelma. Se voidaan jakaa kevyempiin itsenäisesti suoritettaviin osaprosesseihin eli säikeisiin (engl. thread), jotka käyttävät tälle prosessille varattuja resursseja. Yleensä prosessissa samaan aikaan suoritettavat toiminnot jaetaan suoritettavaksi omissa säikeissään, koska ne toimivat synkronisesti. Node.js-sovellus taas suorittaa ohjelmakoodin tapahtumasilmukassa, joka on yhdessä säikeessä. Tapahtumasilmukassa eri ohjelman tapahtumat ohjataan omille niitä käsitteleville asynkronisille funktioilleen. Asynkronisuus mahdollistaa ohjelman suorituksen yhdessä säikeessä, koska asynkroniset funktiot eivät missään vaiheessa estä tapahtumasilmukan etenemistä. Synkroniset, ohjelman etenemisen estävät funktiot ja laskennallisesti raskaammat tehtävät, jotka vievät pidemmän aikaa, suoritetaan tapahtumasilmukasta erillisenä niin sanotun työlaispoolin säikeissä. [20, s.1, 3-4] [21-22]

Node.js soveltuu asynkronisuutensa takia hyvin sovelluksiin, joissa käyttäjän syötteiden perusteella haetaan tai syötetään tietoja esimerkiksi tietokantaan, API-rajapintaan tai tiedostoon. Asynkroninen suoritus mahdollistaa useiden käyttäjän pyyntöjen suorittamisen samanaikaisesti, koska yhden pyynnön valmistumista ei tarvitse odottaa ennen uusien pyyntöjen tekemistä. [20, s. 6-7]

Node.js mahdollistaa JavaScript-ohjelmointikielen käytön sekä palvelin- että selainohjelmassa. Saman ohjelmointikielen käytössä on useita etuja. Palvelin- ja selainpuolen ohjelmointiin voidaan käyttää samoja kehitystyökaluja ja samat ohjelmoijat voivat työskennellä molempien sovelluksen osien parissa. Koodin siirtäminen puolien välillä on helppoa ja molemmat ohjelmat käyttävät suoraan samoja tietotyyppejä, kuten JSON (JavaScript Object Notation). [23, s.4-5]

NPM (Node Package Manager) on Node.js:n pakettien hallintatyökalu. NPM:n kautta on mahdollista ladata, asentaa ja hallinnoida avoimen lähdekoodin ohjelmakirjastoja, jotka laajentavat Node.js:n toiminnallisuutta. NPM:ssä on tarjolla satoja tuhansia paketteja lukuisiin eri käyttötarkoituksiin. [20, s.4] NPM koostuu verkkosivustosta, pakettirekisteristä ja komentorivityökalusta. Verkkosivuston kautta voi etsiä rekisterissä olevia paketteja ja lisätä uusia paketteja rekisteriin tarjolle. Paketteja etsiessä niitä pystyy lajittelemaan muun muassa suosion ja päivitystiheyden perusteella. Pakettikohtaisesti näytetään yhteenveto paketin tiedoista ja komentorivikomento paketin asentamiseen. [22]

Komentorivityökalua käytetään pakettien asentamiseen, päivittämiseen ja poistamiseen. Paketteja voidaan asentaa joko paikallisesti pelkästään yhden Node.js-projektin käyttöön tai globaalisti, jolloin paketti lisätään PATH-ympäristömuuttujaan. Asennettaessa pakettia asennetaan myös sen toimintaan vaadittavat paketit. Node.js-projektille luodaan package.json-tiedosto, joka sisältää listan projektin tarvitsemista paketeista ja niiden versioista. Tämä helpottaa projektin jakamista, koska asennettujen pakettien sijaan riittää, että projektin mukana jaetaan pelkästään package.json-tiedosto. [22]

3.6 NodeOPCUA-kehitystyökalu

NodeOPCUA-kehitystyökalu [24] on Etienne Rossignonin kehittämä Node.js-pohjainen avoimen lähdekoodin OPC UA -kehitystyökalu ja -pino. NodeOPCUA-kehitystyökalun kehityksestä, ylläpidosta ja tukipalveluista vastaa Rossignonin Sterfive-yritys. Sterfive-yritykseltä on mahdollista saada maksullista tukea NodeOPCUA-kehitystyökalun käyttöön ja OPC UA -sovellusten kehitykseen. NodeOPCUA on saatavilla NPM:ssä ja GitHub-sivustolla. Github-sivustolla on myös mahdollista osallistua NodeOPCUA-kehitystyökalun kehitykseen. NodeOPCUA on käytettävissä MIT-lisenssillä eli sitä voi käyttää ja muokata vapaasti myös kaupallisissa sovelluksissa. NodeOPCUA-kehitystyökalua on hyödynnetty muun muassa Node-RED -ympäristön OPC UA -liitettävyyden toteuttamiseen [25].

NodeOPCUA-kehitystyökalua voidaan käyttää sekä OPC UA -asiakassovellusten ja -palvelinsovellusten kehitykseen, joten testaamista varten samaa kehitystyökalua voidaan käyttää testipalvelimen toteuttamiseen. Se tukee tällä hetkellä OPC.TCP-Binary -siirtoprotokollaa ja tärkeimpiä OPC UA -palvelujoukkoja. Lisäksi NodeOPCUA-kehitystyökaluun on tarjolla kattava rajapintadokumentaatio ja useita esimerkkisovelluksia GitHub-sivustolla. OPC UA -asiakassovellusmoduulin kannalta tärkeimmät toiminnot löytyvät jo kehitystyökalusta ja siihen lisätään uusia ominaisuuksia ja parannuksia ja sen bugeja korjataan aktiivisesti.

3.7 Ohjelmakirjastot

OPC UA -asiakassovellusmoduuli toteutetaan paikallisesti Node.js-palvelinohjelmana. Sovelluksen käyttöliittymä toteutetaan verkkosivuna. Node.js-sovelluksen asynkronisuus mahdollistaa käyttöliittymän toimimisen yhden verkkosivun sovelluksena, joka hakee tietoja käyttäjän pyyntöjen perusteella ja päivittyy reaaliaikaisesti. Jako erilliseen palvelinsovellukseen ja verkkosivupohjaiseen käyttöliittymään on monella tapaa hyödyllinen.

Verkkosivu käyttöliittymänä mahdollistaa moduulin itsenäisen käytön ja samalla käyttöliittymä pystytään upottamaan osaksi EloWise- tiedonhallintatyökalua. Lisäksi selainikkunaan avattu käyttöliittymä voidaan sulkea ja palvelinohjelma jatkaa toimintaansa taustalla. Näin esimerkiksi OPC UA -tilausten määritysten jälkeen tiedonkeruu onnistuu ilman käyttöliittymän aukioloa. Verkkosivupohjaisen käyttöliittymän kehityksessä pystytään hyödyntämään HTML (Hypertext Markup Language) -elementtejä ja JavaScript-kirjastoja, jotka helpottavat kehitystä ja lyhentävät kehitykseen vaadittavaa aikaa.

Lokaali palvelinohjelma voidaan tarvittaessa siirtää suoritettavaksi palvelintietokoneelle, jolloin useampi käyttäjä voi esimerkiksi seurata samoja OPC UA -tilauksen arvojen muutoksia omalta laitteeltaan. Tällöin käyttäjän laitteelle ei tarvitse erikseen asentaa sovellusta vaan pelkkä selain riittää sovelluksen käyttämiseen, jolloin sovellusta voidaan käyttää myös mobiililaitteilla.

Palvelinsovelluksen ja käyttöliittymäverkkosivun toteuttamiseen käytetään NPM-pakettirekisterin kautta saatavia avoimen lähdekoodin ohjelmakirjastoja. Valmiiden ohjelmakirjastojen avulla sovelluksen kaikkia toimintoja ei tarvitse toteuttaa alusta alkaen itse.

Toteutukseen on valittu yleisesti käytössä olevat ja toimivaksi todetut ohjelmakirjastot jokaista tarvittavaa toiminnallisuutta varten. Suosituimmat ohjelmakirjastot ovat pitkälle kehitettyjä, niitä päivitetään aktiivisesti ja niiden käyttöön on saatavilla oppaita ja tukea. Valitut ohjelmakirjastot ja niiden toiminta ja käyttötarkoitus ohjelman kehityksessä on esitetty seuraavissa alaluvuissa.

3.7.1 Express-ohjelmistokehys

Express [26] on Node.js-ohjelmistokehys websovellusten kehitystä varten. Express on minimaalinen, mutta joustava ohjelmistokehys. Itsessään se tarjoaa vain websovelluksen perustoiminnallisuudet, mutta toimintoja on mahdollista lisätä sivumallimoottoreilla, laajennusmoduuleilla ja niin sanotuilla väliohjelmistoilla (engl. middleware). Express ei kuitenkaan sido käyttämään tiettyjä laajennuksia, vaan tarjolla on useita vaihtoehtoja, joista voi valita omaan sovellukseen parhaiten soveltuvat vaihtoehdot.

Express-sovelluksessa reititinväliohjelmisto ohjaa pyynnöt osoitteiden perusteella niille määritellyille käsittelijäväliohjelmistoille. Väliohjelmisto on funktio, joka voi esimerkiksi käsitellä saapunutta http-pyyntöä, tehdä toiminnon sen perusteella, lähettää vastauksen pyyntöön tai välittää pyynnön toiselle väliohjelmistolle. Väliohjelmiston parametreinä ovat pyyntöobjekti, vastausobjekti ja mahdollinen viittaus seuraavaan väliohjelmistoon. [27, s.24,43-48]

Verkkosivupohjaisen käyttöliittymän jakaminen ja käyttöliittymän ja paikallisen palvelinsovelluksen kommunikointi pystytään toteuttamaan Express-websovelluksena. Käyttäjän käyttöliittymän kautta lähettämät pyynnöt ohjataan väliohjelmistoille, jotka kutsuvat pyydettyä toimintoa vastaavaa funktiota.

3.7.2 Sequelize-ORM -ohjelmakirjasto

Sequelize [28] on ORM (Object-relational mapping) -ohjelmakirjasto Node.js-kehitysympäristöön. ORM-ohjelmakirjaston avulla tietokantarelaatiot yhdistetään niitä vastaaviin JavaScript-objekteihin, jonka jälkeen tietokantaoperaatioita pystytään suorittamaan ohjelmakoodissa relaatiota vastaavan objektin funktiokutsuilla tietokantakyselyiden kirjoittamisen sijaan. Sequelize tukee useita eri SQL (Structured Query Language) -variaatiota, kuten esimerkiksi MySQL ja MSSQL, joten jos tietokantaa myöhemmin vaihdetaan variaatiosta toiseen, samat ohjelmakoodit toimivat vain Sequelize-ohjelmakirjaston asetuksia muuttamalla.

Sequelize-ohjelmakirjaston avulla sovellus voidaan yhdistää MSSQL-tietokantaan ja sen avulla pystytään luomaan tarvittavat taulut tietokantaan tiedonkeruuta varten. Sen avulla pystytään myös tallentamaan sovelluksen luomien OPC UA -tilausten ja niiden alkioiden parametrejä vikatilanteista palautumisen helpottamiseksi.

3.7.3 Socket.IO-ohjelmakirjasto

Websocket-protokolla on osa HTML5-webtekniikoita. WebSocket-tekniikassa muodostetaan kaksisuuntainen yhteys selaimen ja palvelimen välille. Yhteys avataan yhdellä http-pyynnöllä ja yhteyden muodostamisen jälkeen sitä käytetään molemman suuntaiseen liikenteeseen. Ajoittaisen kyselyn sijaan viestejä lähetetään vain, kun niitä on saatavilla.

Kyselyyn verrattuna WebSocket parantaa suorituskykyä, vähentää viiveitä ja yksinkertaistaa reaaliaikaisen liikenteen toteuttamista. [29, luku 1]

Socket.IO [30] on WebSocket-protokollaa käyttävä ohjelmakirjasto, joka mahdollistaa reaaliaikaisen kommunikoinnin selainpuolen käyttöliittymän ja palvelinpuolen ohjelman välillä. Palvelinpuolella käytetään Node.js-ohjelmakirjastoa ja selaimessa vastaavaa JavaScript-ohjelmakirjastoa. Eri tapahtumien pohjalta lähetetään viestejä selaimen ja palvelinohjelman välillä, jotka sisältävät dataa JSON-muodossa.

Yhdessä Express-ohjelmistokehyksen kanssa Socket.IO:lla pystytään toteuttamaan verkkosivu, jossa esimerkiksi tiedot taulukossa tai kuvaajassa päivittyvät reaaliajassa ilman sivun uudelleen lataamista. Reaaliajassa päivittyvä taulukko soveltuu hyvin OPC UA -tilausten arvojen muutosten seurantaan.

3.7.4 jQuery & Fancytree -ohjelmakirjastot

jQuery [31] on ohjelmakirjasto, joka helpottaa JavaScript-ohjelmointia verkkosivuja varten monin tavoin. jQuery tarjoaa valitsimia ja metodeja HTML-dokumenttiobjektien hallinnointiin ja manipulointiin ja yksinkertaistaa tapahtumien käsittelyä ja sivujen animointia. jQuery:ä käyttämällä pystyy välttämään selainten välisistä JavaScript-eroavaisuuksista johtuvia ongelmatilanteita ja sen tarjoamat valitsimet toimivat myös vanhemmissa selaimissa, jotka eivät itsessään tue HTML dokumenttiobjektien valitsimia. jQuery:n pääperiaatteena on tehdä yleisten toimintojen toteuttamisesta yksinkertaista ja vähentää toteuttamiseen vaadittavaa koodin määrää. [32, luku 1]

Fancytree [33] on jQuery-liitännäinen puunäkymän toteuttamiseen. Sen avulla on mahdollista toteuttaa puurakenne, joka rakennetaan AJAX (Asynchronous JavaScript And XML) -kyselyjen pohjalta ja jossa kansioiden sisältö ladataan niin sanotusti laiskasti eli vasta, kun kansio avataan. Puunäkymä soveltuu hyvin OPC UA -palvelimen osoiteavaruuden visualisointiin ja osoiteavaruuden selauksen toteuttamiseen. Osoiteavaruuden objektit voidaan esittää puunäkymässä kansioina ja vastaavasti objekteihin kuuluvat muuttujat kansioiden sisällä olevina tiedostoina. Kansioiden laiska lataus mahdollistaa OPC UA -palvelimen osoiteavaruuden muutoksiin reagoimisen ilman verkkosivun päivitystä.

3.7.5 Winston-ohjelmakirjasto

Sovellusta käytetään tiedonkeruuta varten taustalla, joten sovelluksen toimintaa ei aina seurata aktiivisesti käyttöliittymästä. Lokikirjausten avulla pystytään jälkeenpäin tarkistamaan sovelluksen toimintaa ja selvittämään mahdollisia sovelluksessa tapahtuneita virhetilanteita ja milloin ne ovat tapahtuneet.

Winston [34] on ohjelmakirjasto lokikirjausten hallinnointiin. Sen avulla pystyy määrittämään lokeille eri kirjaustasoja, tasojen vakavuuksia ja lokien tallennuspaikkoja. Lokikirjauksia voi Winstonin avulla tallentaa konsolin ja lokitiedostojen lisäksi verkkosijainteihin ja tietokantoihin.

4. TOTEUTETTU SOVELLUS

4.1 Kehitysympäristö

Kehitykseen käytettävälle työasemalle asennettiin Node.js-sovellusalusta ja NPM-pakettienhallintatyökalu. NPM:n kautta ladattiin OPC UA -toteutus NodeOPCUA ja luvussa 3.7 esitellyt ohjelmakirjastot sovelluksen työhakemistoon. NodeOPCUA:n ja ohjelmakirjastojen dokumentaatioita ja GitHub-sivustoja käytettiin apuna kehityksessä. Lisäksi työasemalle asennettiin MSSQL ja luotiin tietokanta testaamista varten.

Työasemalle asennettiin myös EloWise-tiedonhallintatyökalu ja EloAI-moduuli aikaisempaan toteutukseen tutustumista varten. Kehitystyön tukena käytettiin myös EloAI-moduulin toimintaa kuvaavaa dokumentaatiota. Kehitystyön edetessä kehitettävän sovelluksen toimintaa vertailtiin toimeksiantajan kanssa aikaisempaan toteutukseen ja varmennettiin, että sovellus toimii kuten halutaan.

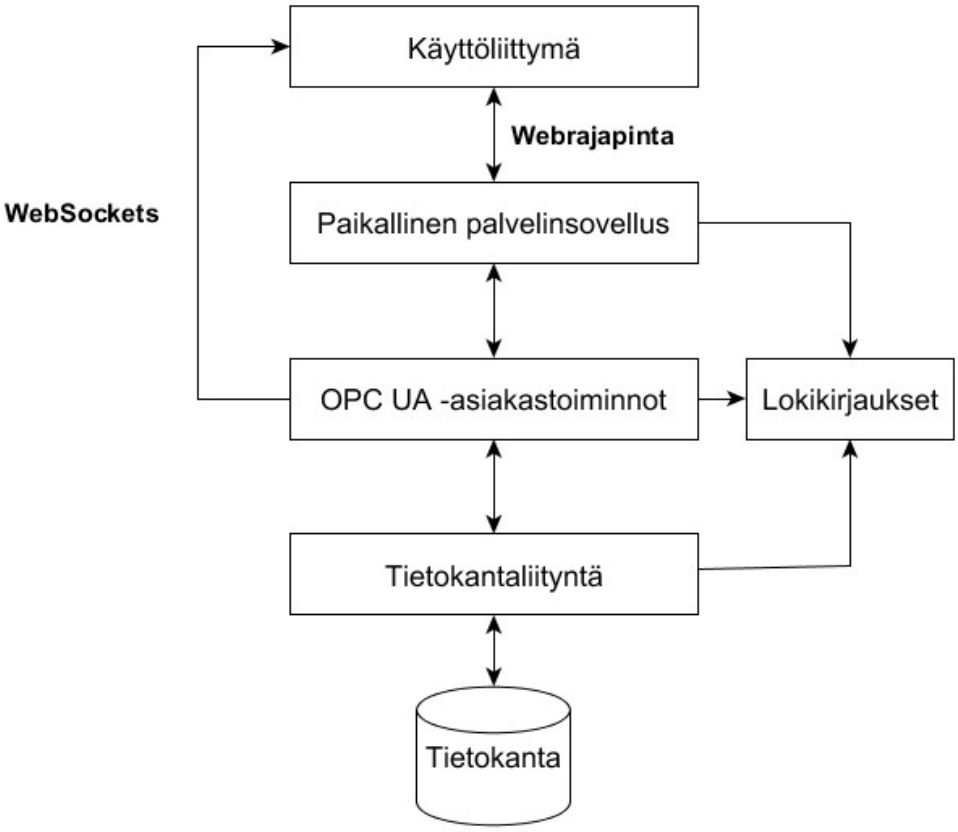
4.2 Sovelluksen rakenne

Valittuja ohjelmakirjastoja käyttämällä kehitettiin OPC UA -asiakassovellusmoduuli. Kehitetty sovellus on paikallisesti suoritettava Node.js-palvelinohjelma, jota käytetään verkkoselaimessa käyttöliittymäverkkosivun tai suoraan webrajapinnan kautta. Sovellusta voidaan käyttää itsenäisenä sovelluksena tai upotettuna osana EloWise-tiedonhallintatyökalua.

Sovellus on jaettu viiteen osakokonaisuuteen. Osakokonaisuudet ovat:

1. paikallinen palvelinsovellus
2. OPC UA -asiakastoiminnot
3. tietokantaliityntä
4. lokikirjaukset
5. käyttöliittymä

Sovelluksen osakokonaisuuksien välisiä riippuvuussuhteita ja tiedonsiirtomenetelmiä on havainnollistettu kuvassa 6.



Kuva 6: Havainnekuva ohjelman rakenteesta

Osakokonaisuudet ovat käyttöliittymää lukuun ottamatta omia Node.js-ohjelmamoduulejaan. Käyttöliittymä koostuu yhdestä verkkosivusta ja siihen liittyvästä JavaScript-ohjelmakoodista. Ohjelman varsinainen suoritettava osuus on palvelinsovellus, joka kutsuu muiden ohjelmamoduulien tarjoamia funktioita ja jakaa käyttöliittymän verkkosivun saataville. Käyttöliittymä voidaan sulkea, jolloin palvelinsovellus jää toimimaan taustalle tiedonkeruuta varten. Osakokonaisuudet ja niiden toiminta on esitetty tarkemmin seuraavissa alaluvuissa.

4.2.1 Palvelinsovellus

Express-ohjelmistokehyksellä toteutettu palvelinsovellus vastaa webrajapinnan toiminnasta ja käyttöliittymänä toimivan verkkosivun jakamisesta. Käyttöliittymän ja palvelinohjelman välinen liikenne on toteutettu webrajapinnan avulla.

Webrajapintaan on toteutettu osoitteet toteutettuja OPC UA -asiakastoimintoja varten. Käyttöliittymäsivun eri painikkeet lähettävät käyttäjän syöttämät tiedot JSON-muodossa painiketta vastaavan toiminnon osoitteeseen. Web-rajapinnan kautta lähetetyssä http-pyynnössä vastaanotetut parametrit tarkistetaan palvelinsovelluksessa, joka kutsuu rajapintakutsua vastaavaa OPC UA -asiakastoiminnon funktiota käyttäjän antamilla parametreilla. Vastauksena palautetaan tilaviesti ja OPC UA -asiakasfunktion palauttamat arvot

JSON-muodossa. Käyttöliittymässä JSON-muotoiset arvot muotoillaan käyttäjälle näkyvään muotoon.

Lisäksi palvelinsovelluksessa alustetaan WebSockets-yhteys ohjelman ja käyttöliittymän välille. Muodostettu yhteys välitetään OPC UA -asiakastoiminnoista vastaavalle osakokonaisuudelle tilausten arvojen reaaliaikaista päivittämistä varten.

4.2.2 OPC UA -asiakastoiminnot

OPC UA -asiakastoiminnot on toteutettu NodeOPCUA-kehitystyökalua käyttäen. OPC UA -asiakastoimintoja varten on luotu funktiot, joiden parametreina annetaan vastaavan OPC UA -palvelun parametrit. Palvelinsovellus kutsuu OPC UA-asiakastoimintojen funktioita käyttäjän antamilla parametreilla ja funktioiden valmistuttua takaisinkutsuna palvelinsovellukselle palautetaan tilakoodi ja -viesti ja kutsutun OPC UA -palvelun palauttamat arvot tai virheilmoitus webrajapintaa varten. Toteutetut funktiot ja kuvaukset niiden parametreista ja toiminnasta on esitetty taulukossa 4.

Taulukko 5: Toteutetut OPC UA -asiakastoiminnot

Funktio	**Parametrit**	**Kuvaus toiminnasta**
InitializeSession		Yhdistäminen tietokantaan ja OPC UA -palvelimeen, luodaan tietokantarelaatiot ja istunto
CloseAndDisconnect		Istunnon ja yhteyden OPC UA -palvelimeen sulkeminen
Browse	Solmujen selausnimistä (BrowseName) koostuva selauspolku	Nimiavaruuden selaus juurikansiosta (RootFolder) alkaen
ReadAttributes	Luettava solmu	Solmun kaikkien attribuuttien lukeminen
ReadValues	Lista luettavista noodeista	Listattujen solmujen Value-attribuuttien lukeminen
Write	Solmu, kirjoitettava arvo, arvon tietotyyppi	Solmun arvon kirjoittaminen

Monitor	Tietokannan parametrit, tilausparametrit, seurattavien alkioiden parametrit	Tilauksen ja seurattavien alkioiden luominen ja tietojen tallennuksen ja seuraamisen aloitus
ReadOnTrigger	Tietokannan parametrit, tilausparametrit, seurattavien alkioiden parametrit	Liipaisupohjaisen tilauksen ja seurattavaien alkioiden luominen ja tietojen tallennuksen ja seuraamisen aloitus
PollingRead	Tietokannan parametrit, tilausparametrit, seurattavien alkioiden parametrit	Kyselypohjaisen tilauksen ja seurattavien alkioiden luominen ja tietojen tallennuksen ja seuraamisen aloitus
RemoveMonitoredItem	Tilauksen nimi, seurattavan solmun nimi	Seurattavan alkion poistaminen annetusta tilauksesta
RemoveSubscription	Tilauksen nimi	Tilauksen poistaminen
ModifySubscription	Tilauksen nimi, tilauksen parametri	Tilauksen parametrien muokkaaminen

Lukufunktioita on kaksi variaatiota. ReadAttributes-funktio lukee yhden solmun kaikki attribuutit. ReadValues-funktio lukee kaikkien annettujen solmujen Value-attribuutin arvon ja palauttaa lisäksi luettujen arvojen tietotyypin, tilan ja aikaleimat arvon päivittämisajasta OPC UA -palvelimella ja siihen yhdistetyllä tietolähteellä.

Browse-funktiolle annetaan parametriksi selauspolku osoiteavaruuden objektiin tai muuttujaan. Funktio palauttaa annetun objektin tai muuttujan sisältämien objektien ja muuttujien polut. Palautettujen polkujen avulla on mahdollista selata osoiteavaruutta eteenpäin.

Write-funktiolla kirjoitetaan valitun solmun Value-attribuutin arvo. Koska parametrit tulevat verkkorajapinnan kautta JSON-muodossa, tarvitaan parametriksi myös arvon tietotyyppi, jotta arvo saadaan muutettua oikean tyyppiseksi ennen kirjoituspyynnön lähettämistä OPC UA -palvelimelle.

Tilausfunktioita on kolme erilaista funktiota eri käyttötapauksia varten. Monitor-funktiota käytetään tavallisten OPC UA -tilausten ja seurattavien alkioiden luomiseen ja seuraamiseen.

NodeOPCUA-kehitystyökaluun ei ollut vielä sovellusta tehdessä toteutettu MonitoredItem-palvelujoukon SetTriggering-palvelua, joka olisi vaadittu liipaisupohjaisia tilauksia varten. Liipaisupohjaisessa tilauksessa seurattavien alkioiden arvot päivitetään vain, kun valitun liipaisualkion arvo muuttuu. Liipaisupohjainen tilaus jouduttiin puuttuvan palvelun takia toteuttamaan omana erillisenä funktionaan, jossa luodaan liipaisualkiolle oma tilauksensa ja aina liipaisualkion muuttuessa luetaan tilauksen muiden seurattavien alkioiden arvot ReadValues-funktiota käyttäen.

Sovellukseen toteutettiin myös syklinen tilaus, jossa tilauksen seurattavien alkioiden arvot päivitetään aina tietyin aikavälein riippumatta siitä, onko arvo muuttunut vai ei. Syklistä lukua ei OPC UA -tilauksen avulla pysty suoraan toteuttamaan, koska OPC UA -tilausten ajatuksena on, että palvelin välittää asiakkaalle vain muuttuneet arvot ylimääräisen liikenteen välttämiseksi. Kuitenkin on käyttötapauksia, joissa seurattavien alkioiden arvoista halutaan kirjaukset tietyn aikavälein arvon muutoksista riippumatta. Tämän takia luotiin oma PollingRead-funktio, jossa intervallifunktion ja toteutetun ReadValues-funktion avulla suoritetaan arvojen luku toistuvasti tietyin aikavälein.

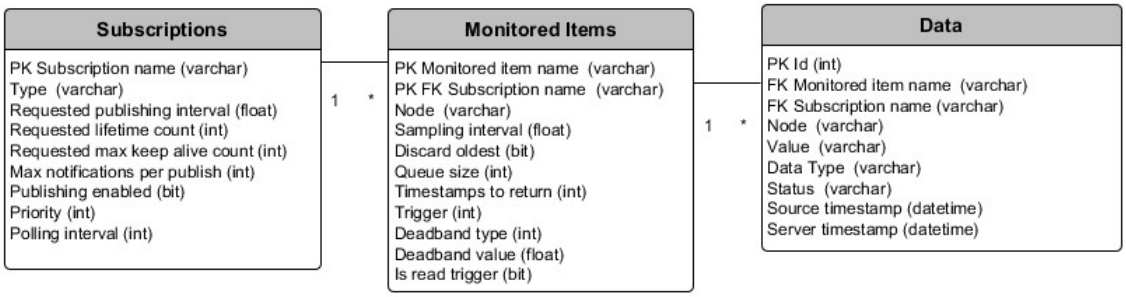
Jokaisessa tilausfunktiossa tilauksen luonnin jälkeen tilaus ja sen seurattavat alkiot tallennetaan ohjelman sisäiseen tietorakenteeseen ja niiden parametrit tallennetaan tietokantaan. Tilauksen seurattavien alkioiden arvojen muutokset tallennetaan tietokantaan ja lähetetään Socket.IO-ohjelmakirjaston avulla selaimeen. Eri tilaustyyppejä poistetaan ja muokataan kuitenkin samoilla funktioilla. Muokattaessa tilausta ModifySubscription-funktiolla tilaus päivitetään OPC UA –palvelimen lisäksi ohjelman sisäisessä tietorakenteessa ja tietokannassa. Tilauksia ja seurattavia alkioita poistettaessa poistetaan ne OPC UA -palvelimelta poiston jälkeen ohjelman sisäisestä tietorakenteesta sekä tietokannasta. Tilauksen kaikki seurattavat alkiot poistetaan sen poistamisen yhteydessä.

Webrajapinnan tarjoamien toimintojen vastineeksi luotujen funktioiden lisäksi on toteutettu funktio OPC UA -palvelimeen yhdistämiseen ja istunnon muodostamiseen, jota kutsutaan sovelluksen käynnistyessä ja vastaavasti funktio yhteyden ja istunnon sulkemiseen, jota kutsutaan, kun sovellus suljetaan. Lisäksi NodeOPCUA-toteutuksen asetuksilla on määritetty, että yhteyden muodostuksen epäonnistuessa tai yhteyden katketessa yhdistämistä yritetään tietyn väliajoin uudestaan.

4.2.3 Tietokantaliityntä

Tietokantaliityntä on toteutettu Sequelize-ohjelmakirjastolla. Tietokantaliityntä vastaa tietokantaan yhdistämisestä, tarvittavien tietokantataulujen luomisesta ja liittämisestä JavaScript-objekteihin ja tietokantaoperaatioiden suorittamisesta. Sovelluksen käynnistyksen yhteydessä yhdistetään asetuksissa määritettyyn MSSQL-tietokantaan ja luodaan tarvittavat tietokantarelaatiot, jos niitä ei vielä ole tietokannassa. Tietokantarelaatioille luodaan ohjelmaan vastaavat objektit, joita käytetään OPC UA -asiakastoimintojen yhteydessä tietojen tallentamiseen tietokantaan.

Tietokantaan luodaan taulut aktiivisten tilausten parametreille, tilausten seurattavien alkioiden parametreille ja tilausten keräämälle datalle. Spesifikaation määrittämien parametrien lisäksi tilaustauluun on lisätty oma kolumninsa kyselypohjaisen tilauksen intervallia varten. Lisäksi seurattavien alkioiden parametreihin on lisätty tieto, onko alkio liipaisualkio vai ei. Luotavat tietokantarelaatiot on esitetty kuvassa 7. Kun tilaus poistetaan sovelluksessa, poistetaan tilauksen ja sen alkioiden parametrit myös tietokannasta.



Kuva 7: Sovelluksen tietokantarelaatiot

Tilausten data voidaan kerätä yhteen tauluun tai tilauskohtaisiin tauluihin, jossa kukin alkio on omana sarakkeenaan. Datatauluissa varsinaiset arvot ovat tekstimuodossa ja tietotyypille on oma sarakkeensa, jotta samaan sarakkeeseen voidaan tallentaa eri tietotyyppien arvoja. Datataulujen sisältöä ei tyhjennetä tilausten poiston yhteydessä. Monimutkaisempaa jatkokäsittelyä varten on myös mahdollista välittää tiedot taulukkona tietokantaproseduureille.

Tietokantaliityntään on lisäksi luotu palautusfunktio, jolla haetaan tietokannasta kaikkien tilausten ja niiden seurattavien alkioiden parametrit. Funktiota kutsutaan ohjelman käynnistyessä ja haettuja parametreja käytetään tilausten ja seurattavien alkioiden luontiin. Palautusfunktion avulla saadaan luotua tilaukset uudelleen ohjelman kaataneen vikatilanteen jälkeen tai tilanteessa, jossa parametritaulut on otettu talteen ja kopioitu uuteen tietokantaan ja sovellus käynnistetään ensimmäistä kertaa.

4.2.4 Lokikirjaukset

Lokikirjausten määrittelyt ja lokikirjaajan alustus on tehty omaan ohjelmamoduuliinsa. Samaa lokikirjaajaa kutsutaan muissa ohjelmamoduuleissa. Lokit tallennetaan tekstitiedostoon, johon kirjataan viesti tai virheilmoitus ja minkä ohjelmamoduulin ja funktion suorituksessa kirjaus tehtiin.

Lokikirjauksia tallennetaan kolmella eriasteisella vakavuustasolla. Infotason kirjauksia tehdään sovelluksen normaalin toiminnan yhteydessä, esimerkiksi, kun yhteys OPC UA -palvelimeen saadaan muodostettua tai kun yhteys katkeaa ja aloitetaan yhdistäminen uudelleen. Varoitustason kirjauksia tehdään, kun toiminnon suoritus estetään ennen virhettä, esimerkiksi kun yritetään poistaa tilaus, jota ei enää ole olemassa. Virhetason kirjauksia

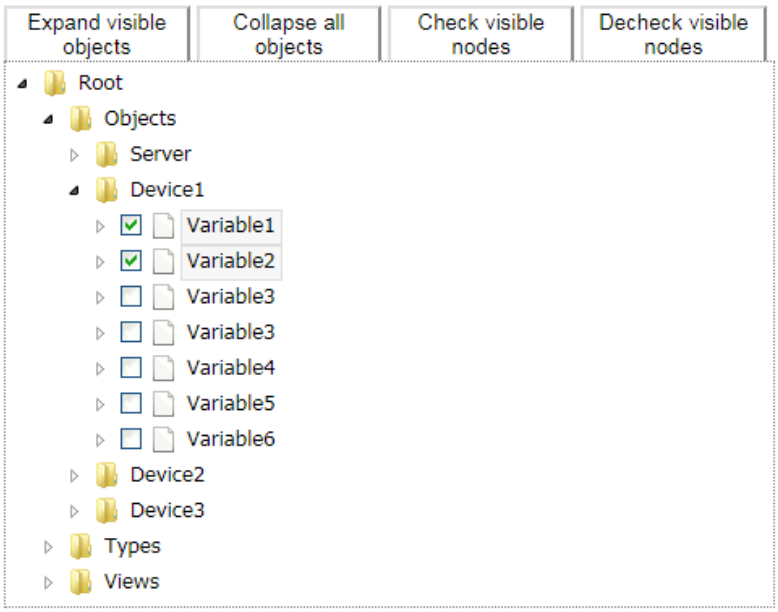
tehdään, kun OPC UA -palvelin palauttaa palvelukutsuun virheen tai ohjelman suorituksessa tapahtuu muu odottamaton virhetilanne.

4.2.5 Käyttöliittymä

Käyttöliittymä on yhden verkkosivun sovellus, jossa näytetään OPC UA -palvelimen nimiavaruus puurakenteena ja puurakenteen rinnalla eri toiminnot omina välilehtinään. Käyttöliittymän toteutukseen on käytetty jQuery-, Fancytree- ja Socket.IO-ohjelmakirjastoja. Käyttöliittymän kautta on mahdollista selata OPC UA -palvelimen nimiavaruutta, lukea ja kirjoittaa solmujen attribuutteja ja arvoja ja luoda ja seurata OPC UA -tilauksia. Käyttöliittymä vastaa käyttäjän syötteiden muuttamisesta oikean muotoisiksi parametreiksi ja niiden välittämisestä webrajapinnalle.

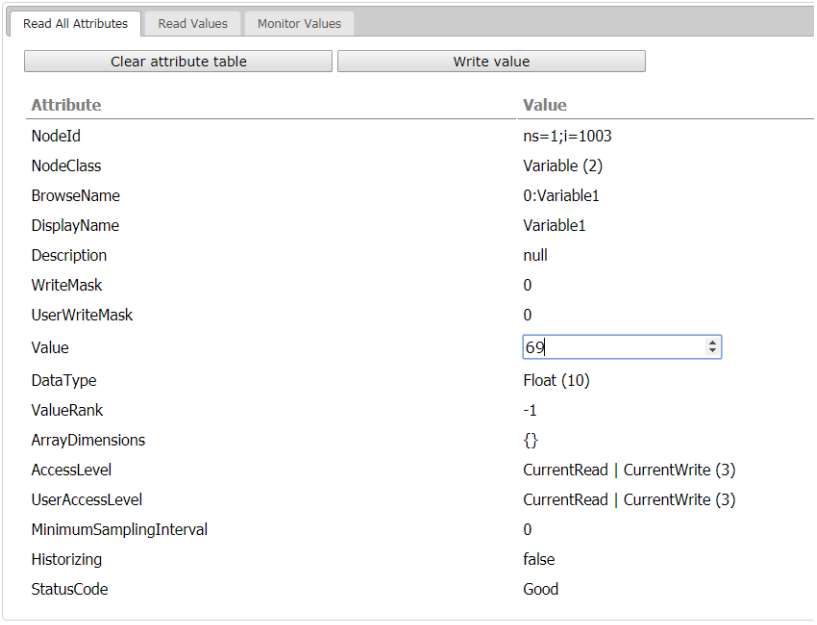
Puurakenteessa OPC UA -objektit on esitetty kansioina ja OPC UA -muuttujat tiedostoina. Puurakenteen sisältö ladataan webrajapinnasta laiskasti eli vasta, kun puun kansioita avataan. Ensimmäinen kansio avaa osoiteavaruuden juurikansion. Juurikansiosta eteenpäin jokaista objektia avatessa lähetetään uusi pyyntö webrajapinnan Browse-palvelulle objektin sisältämien objektien ja muuttujien selvittämiseksi. Laiskan latauksen ansiosta koko osoiteavaruutta ei avata samanaikaisesti, mikä saattaisi viedä laajalla osoiteavaruudella huomattavasti aikaa. Lisäksi osoiteavaruuden vaiheittainen selaamisen ansiosta nimiavaruuden muutokset päivittyvät puurakenteeseen, kun objekteja avataan sen sijaan, että koko puurakenteen sisältö ladattaisiin vain aluksi.

Puurakenteen muuttujia on mahdollista merkitä klikkaamalla lukemisen ja tilausten tekemisen helpottamista varten. Lisäksi puurakenteen yläpuolella on pikanäppäimiä puun laajentamiseen ja sulkemiseen ja useiden muuttujien merkitsemiseen. Puurakenne on esitetty kuvassa 8.



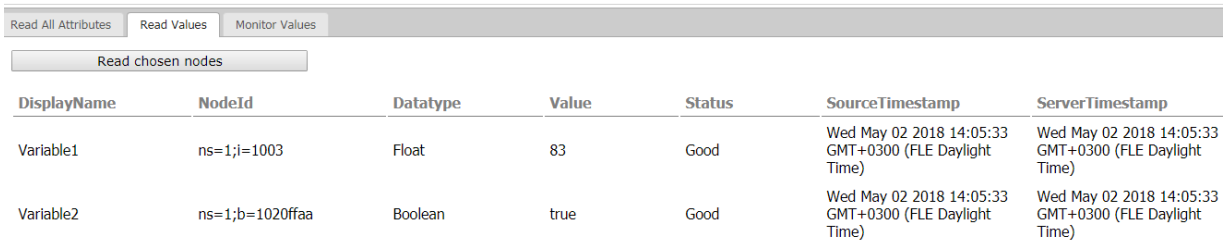
Kuva 8: Käyttöliittymän puurakenne

OPC UA -palvelimen osoiteavaruutta selatessa näytetään puurakenteesta valitun solmun kaikki attribuutit Read All Attributes -välilehdellä. Välilehdellä näytetään selatun objektin tai solmun kaikki attribuutit ja niiden arvot. Välilehden arvot päivittyvät, kun puusta valitaan uusi objekti tai solmu. Välilehdellä voidaan myös muokata solmun Value-attribuutin arvoa, jos solmuun on kirjoitusoikeus. Kirjoittaminen on mahdollista yksinkertaisille tietotyypeille, jotka pystytään syöttämään teksti- tai numerokenttään. Käyttäjän syöttämä arvo tarkistetaan ja ainoastaan kelvollisen arvon kirjoittaminen on mahdollista. Kirjoittamisen onnistumisesta näytetään käyttäjälle ilmoitus ja välilehdellä näytetyt arvot päivitetään. Read All Attributes -välilehti on esitetty kuvassa 9.



Kuva 9: Read All Attributes -välilehti

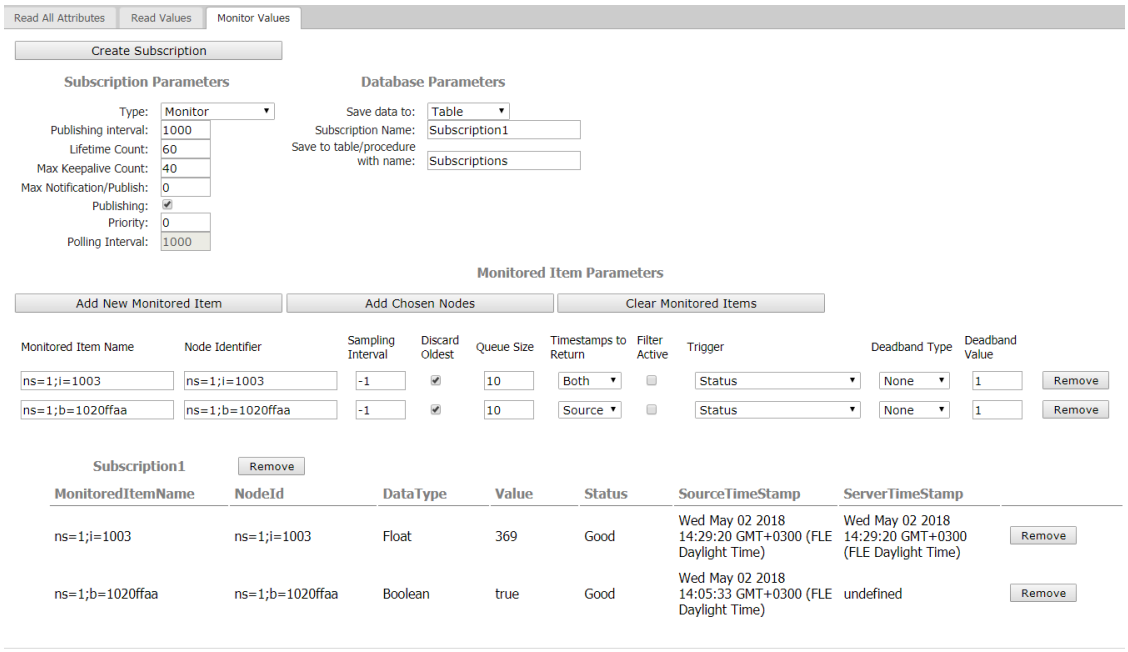
Read Values -välilehdellä voidaan lukea puurakenteesta merkittyjen solmujen Value-attribuutin arvot ja kerätä ne taulukkoon. Varsinaisen arvon lisäksi taulukkoon listataan luetun solmun nimi ja tunnus, luetun arvon tietotyyppi ja tila ja aikaleimat arvon päivittämisajasta OPC UA -palvelimella ja siihen yhdistetyllä tietolähteellä. Taulukon arvot voidaan päivittää lukemalla tiedot uudestaan. Read Values- välilehti on kuvassa 10.



Kuva 10: Read Values -välilehti

Monitor Values -välilehdellä määritetään ja luodaan OPC UA -tilauksia. Välilehdellä on omat parametrinsa tilaukselle, seurattaville alkioille ja tiedonkeruulle. Käyttäjä voi valita tilaukselle ja sen seurattaville alkioille nimet, joita käytetään niiden tunnisteina käyttöliittymässä ja tietokannassa.

Tilauksen tyyppi voidaan valita kolmesta luvussa 4.2.2 esitetystä vaihtoehdosta, jonka jälkeen syötettäviä parametreja rajataan valitun vaihtoehdon mukaan. Tilauksen seurattavat alkiot voidaan valita puurakenteesta merkityistä solmuista tai vaihtoehtoisesti voidaan syöttää seurattavien solmujen tunnisteet suoraan lomakkeeseen. Puurakenteesta valittujen solmujen nimenä käytetään oletuksena niiden omaa tunnistetta (nodeId). Tilauskohtaisesti voidaan valita, tallennetaanko tilauksen arvot tietokannan tauluun vai tietokantaproseduuriin. Monitor Values -välilehti on kuvassa 11.



Kuva 11: Monitor Values -välilehti

Käyttäjän syöttämillä parametreilla luodaan tilaus ja tilauksen seurattavat alkiot. Luodun tilauksen ja sen seurattavien alkioiden parametrit tallennetaan tietokantaan omiin tauluihinsa. Tilauksen luomisen jälkeen tilauksen seurattavien alkioiden arvoja kerätään valittuun tietokannan tauluun tai tietokantaproseduuriin.

Seurattavien alkioiden arvot näytetään lisäksi käyttöliittymässä reaaliaikaisesti päivittyvässä taulukossa, jonka kautta on myös mahdollista poistaa seurattavia alkioita tilauksesta. Luodut tilaukset näytetään omina taulukkoinaan allekkain luontijärjestyksessä ja niiden parametrejä pystyy päivittämään tilauksen parametrien lomaketta käyttäen. Tilauksen ja sen alkioiden poistoon on omat painikkeensa taulukon yhteydessä.

4.3 Sovelluksen testaus

Sovelluksen testaamista varten toteutettiin yksinkertainen OPC UA -palvelinsovellus NodeOPCUA-kehitystyökalulla. NodeOPCUA-kehitystyökalun GitHub-sivustolla jaettavaa esimerkkipalvelinsovellusta käytettiin pohjana testipalvelinsovelluksen toteutukselle. Testipalvelinsovelluksen avulla asiakassovelluksen toimintaa saatiin testattua ja varmennettua kehityksen edetessä. Testipalvelinsovellusta ajettiin asiakassovelluksen kanssa samalla työasemalla. Saman kehitysympäristön käyttäminen testipalvelinsovellusta varten ja asiakas- ja palvelinsovelluksen ajaminen samalla tietokoneella yksinkertaisti ja nopeutti testausta.

Testipalvelimelle luotiin esimerkkiobjekteja ja eri tietotyyppien muuttujia. Asiakassovelluksella yhdistettiin testipalvelinsovelluksen alustamaan OPC UA -palvelimeen ja luettiin ja kirjoitettiin muuttujien arvoja ja luotiin tilauksia ja testattiin eri tilausten parametrien

toimintaa. Lisäksi asiakassovelluksen toimintaa verrattiin ilmaisen CommServer OPC UA Viewer -sovelluksen toimintaan samalle testipalvelimelle yhdistettäessä.

Testipalvelimeen kanssa testaamisen lisäksi sovellusta testattiin toimintojen valmistuttua toimeksiantajan testitilan laitteiden kanssa. Sovellus yhdistettiin lähiverkon välityksellä Siemensin PCS7-logiikan yhteyteen luotuun OPC UA -palvelimeen. Sovelluksella selattiin palvelimen osoiteavaruutta ja palvelimen kautta luettiin ja seurattiin PCS7-logiikan muistialueelta tiedostoyksikön muuttujia.

5. YHTEENVETO

5.1 Työn tulokset

EloWise-tiedonhallintatyökaluun oli aikaisemmin toteutettu OPC DA -asiakassovellusmoduuli, josta haluttiin siirtyä OPC DA:n vanhenemisen takia OPC UA:han. Diplomityön tavoitteena oli perehtyä OPC-tekniikoihin ja eri siirtymismenetelmiin OPC Classic -tekniikoista OPC UA:han, esitellä ja vertailla saatavilla olevia OPC UA -kehitystyökaluja ja -pinoja sekä päivittää EloWise-tiedonhallintatyökalu OPC UA -liitettävyyttä varten.

Aluksi vertailtiin eri menetelmiä OPC Classic -tekniikoista OPC UA:han siirtymiseksi. Vaihtoehtoina oli kääre- ja välityskomponenttien tai muunnosten käyttäminen, vanhan OPC DA -asiakassovellusmoduulin päivittäminen käyttämään OPC UA:ta tai kokonaan uuden OPC UA -asiakassovellusmoduulin kehittäminen. Siirtymismenetelmäksi valittiin uuden OPC UA -asiakassovellusmoduulin kehittäminen, jotta aikaisemman moduulin toteutus ei rajoittanut kehitystä ja pystyttiin hyödyntämään kaikkia OPC UA:n ominaisuuksia. Vaatimuksiksi uudelle moduulille asetettiin aikaisempaa OPC DA -asiakassovellusmoduulia vastaavat toiminnot.

Kehityksessä käytettävän OPC UA -kehitystyökalun ja -pinon valintaa varten vertailtiin saatavilla olevia OPC UA -kehitystyökaluja ja -pinoja. Saatavilla oli useita kaupallisia ja avoimen lähdekoodin kehitystyökaluvaihtoehtoja eri ohjelmointikielille ja kehitysympäristöille. Yksinkertaisten vaatimusten ja kehityskustannusten minimoimiseksi päätettiin käyttää avoimen lähdekoodin kehitystyökalua. Avoimen lähdekoodin kehitystyökaluista valittiin Node.js-pohjainen NodeOPCUA. NodeOPCUA-kehitystyökalussa oli kaupalliseen sovellukseen soveltuva salliva MIT-lisenssi, OPC UA -asiakassovellusmoduulin toteutukseen kannalta tarvittavat toiminnot, sitä kehitettiin aktiivisesti ja siihen oli saatavilla kattava dokumentaatio. Lisäksi Node.js-kehitysympäristö oli ennestään tuttu ja sille oli saatavilla useita kehitystyötä helpottavia valmiita ohjelmakirjastoja eri ominaisuuksien toteuttamiseen. Sovelluksen kehitykseen valittiin joukko muita yleisesti käytettyjä avoimen lähdekoodin ohjelmakirjastoja. Express-ohjelmistokehystä käytettiin websovelluksen toimintoihin, Sequelize-ORM-ohjelmakirjastoa tietokantaliityntää varten, Socket.IO-ohjelmakirjastoa käyttöliittymän tietojen reaaliaikaiseen päivittämiseen, jQuery- ja Fancytree- ohjelmakirjastoja käyttöliittymän toimintoihin ja Winston-ohjelmakirjastoa lokikirjausten hallinnointiin.

Valitulla kehitystyökaluilla kehitettiin OPC UA -asiakassovellusmoduuli, joka koostuu paikallisesta palvelinsovelluksesta ja selaimessa käytettävästä käyttöliittymästä. Sovellusta voidaan käyttää itsenäisesti tai modulaarisena osana EloWise-tiedonhallintatyöka-

lua. Sovelluksen avulla on mahdollista yhdistää OPC UA -palvelimiin, selata niiden osoiteavaruutta ja lukea ja kirjoittaa muuttujien arvoja. Sovelluksen avulla voidaan lisäksi seurata tilauksia reaaliaikaisesti ja kerätä niiden arvoja tietokantaan.

Uuden OPC UA -asiakassovelluksen kehitys osoittautui toimivaksi menetelmäksi OPC DA:sta OPC UA:han siirtymiseen. OPC UA -kehitystyökalu ja ohjelmakirjastot helpottivat toteutusta ja sovelluksen toteutus onnistui hyvin sille varatussa ajassa. Valittu kehitystyökalu oli riittävä sovelluksen toteutukseen. Kehitystyökalussa oli kuitenkin joitakin puutteita ja sen käytössä ilmeni ongelmatilanteita, joihin sai heikosti apua kehitystyökalun ylläpitäjältä. Ongelmatilanteista selvittiin ja kehitystyökalua päivitetään yhä aktiivisesti, joten puutteet tulevat korjatuksi tulevaisuudessa.

Diplomityön tuloksena on kirjallisuuskatsaus OPC -tekniikoihin, menetelmiin OPC-Classic -tekniikoista OPC UA:han siirtymiseen ja saatavilla oleviin OPC UA -kehitystyökaluihin ja -pinoihin. Lisäksi diplomityössä toteutettiin ja esiteltiin esimerkki OPC UA:han siirtymisestä kehittämällä uusi OPC UA -asiakassovellus. Samalla esiteltiin kehityksessä käytetty kehitysympäristö, OPC UA -toteutus ja muita työn kehityksessä hyödynnettyjä ohjelmakirjastoja. Diplomityön tuloksia voidaan käyttää apuna siirtymistä OPC Classic -tekniikoista OPC UA:han toteutettaessa tai uutta OPC UA -asiakassovellusta kehittäessä.

5.2 Jatkokehitys

Toteutettu sovellus on asetetut vaatimukset täyttävä, mutta nykyisiltä ominaisuuksiltaan suppea ja sen käyttöliittymä on pelkistetty. Toteutus on kuitenkin hyvä pohja jatkokehitykselle ja sovellukseen on jatkossa helppo lisätä uusia ominaisuuksia. Tulevaisuudessa on tarkoitus jatkaa sovelluksen kehitystä ja lisätä siihen uusia ominaisuuksia ja päivittää sovelluksen käyttöliittymän ulkoasua. Sovellusta päivitetään sitä mukaa, kun käytettyyn OPC UA -toteutukseen tulee uusia ominaisuuksia ja toimintoja ja käyttöliittymään lisätään kuvaajia ja muita keinoja kerättyjen tietojen visualisointiin.

Liipaisupohjainen tilaus muutetaan väliaikaisesta ratkaisusta käyttämään OPC UA -spesifikaation mukaista SetTriggering-palvelua, kun toiminto tulee saataville NodeOPCUA-kehitystyökaluun. Sovellukseen voidaan lisäksi toteuttaa jo kehitystyökalussa olevat metodien kutsuminen ja Discovery-palvelujoukon toiminnot palvelimien etsintään. Samalla voidaan lisätä tuki useampaan palvelimeen yhdistämiseen samanaikaisesti ja oma käyttöliittymävälilehtensä palvelimien hallinnoinnille.

Jatkossa asiakasohjelmaa voitaisiin käyttää paikallisen palvelimen sijaan keskitetyllä palvelimella, jolloin useampi käyttäjä voisi käyttää sovellusta oman selaimensa kautta ilman paikallista ohjelman suoritusta. Tällöin ainoastaan keskitetyllä palvelimella täytyisi olla yhteys seurattavaan OPC UA -palvelimeen. Keskitettyä käyttöä varten ohjelmaan tulisi lisätä käyttäjien tunnistus ja eri käyttöoikeustasoja, jotta useampi käyttäjä voisi käyttää samaa yhteyttä OPC UA -palvelimeen ilman ristiriitoja. Hallinnointioikeudet omaava

käyttäjä voisi kirjoittaa OPC UA -palvelimen muuttujien arvoja ja luoda ja muokata tilauksia, kun taas tavalliset käyttäjät voisivat vain selata OPC UA -palvelimen sisältöä, lukea arvoja ja seurata luotujen tilausten arvojen muutoksia.

Käyttöliittymään on suunniteltu lisättäväksi erilaisia visualisointeja tilausten keräämää dataa varten. Datasta voidaan piirtää esimerkiksi kuvaajia, tai arvoja voidaan näyttää erilaisissa mittareissa. Kuvaajat ja mittarit voidaan koota muokattavaan kojelautaan, josta näkee nopeasti seurattavien tilausten tilanteen. Datan visualisointiin on useita avoimen lähdekoodin JavaScript-ohjelmakirjastoja, joilla visualisoinnit pystyttäisiin toteuttamaan nopeasti.

LÄHTEET

[1] T. Hannelius, M. Salmenperä, S. Kuikka, Roadmap to adopting OPC UA, 2008 6th IEEE International Conference on Industrial Informatics, IEEE, pp. 756-761.

[2] B. Farnham, R. Barillere, Migration from OPC-DA to OPC-UA, Contributions to the Proceedings of ICALEPCS 2011, Ranska, pp. 1423.

[3] Y. Chuanying, L. He, L. Zhihong, Implementation of migrations from Class OPC to OPC UA for data acquisition system, 2012 International Conference on System Science and Engineering (ICSSE), IEEE, pp. 588-592.

[4] H. Haskamp, M. Meyer, R. Mollmann, F. Orth, A.W. Colombo, Benchmarking of existing OPC UA implementations for Industrie 4.0-compliant digitalization solutions, 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), IEEE, pp. 589-594.

[5] F. Palm, S. Gruner, J. Pfrommer, M. Graube, L. Urbas, Open source as enabler for OPC UA in industrial automation, 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, pp. 1-6.

[6] S. Profanter, K. Dorofeev, A. Zoitl, A. Knoll, OPC UA for plug & produce: Automatic device discovery using LDS-ME, 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, pp. 1-8.

[7] D. Schulz, R. Braun, J. Schmitt, Behind the façade, 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, pp. 1-8.

[8] J. Lange, F. Iwanitz, OPC — Openness, Productivity, and Connectivity, in: R. Zurawski (ed.), Integration Technologies for Industrial Automated Systems, CRC Taylor & Francis, Boca Raton, FL, 2006, pp. 30.

[9] M. Damm, S. Leitner, W. Mahnke, OPC Unified Architecture, Springer-Verlag, Berliini, Heidelberg, 2009, 339 p.

[10] Certification: Overview and Benefits, OPC Foundation. Saatavissa (viitattu 01.05.2018): https://opcfoundation.org/certification/overview-benefits/.

[11] Products, OPC Foundation. Saatavissa (viitattu 01.05.2018): https://opcfoundation.org/products.

[12] The Component Object Model, Microsoft. Saatavissa (viitattu 17.07.2018): https://msdn.microsoft.com/en-us/library/windows/desktop/ms694363(v=vs.85).aspx.

[13] J.S. Rinaldi, OPC UA–Unified Architecture: The Everyman's Guide to the Most Important Information Technology in Industrial Automation, CreateSpace Independent Publishing, Yhdysvallat, 2016, 161 p.

[14] Unified Architecture, OPC Foundation. Saatavissa (viitattu 01.05.2018): https://opcfoundation.org/about/opc-technologies/opc-ua/.

[15] Specifications, OPC Foundation. Saatavissa (viitattu 05.06.2018): https://opcfoundation.org/developer-tools/specifications-unified-architecture.

[16] OPC UA .NET StandardLibrary Stack and sample applications, OPC Foundation. Saatavissa (viitattu 01.05.2018): http://opcfoundation.github.io/UA-.NETStandard/.

[17] List of Open Source OPC UA Implementations, GitHub. Saatavissa (viitattu 01.05.2018): https://github.com/open62541/open62541/wiki/List-of-Open-Source-OPC-UA-Implementations.

[18] Licenses, GitHub. Saatavissa (viitattu 04.07.2018): https://choosealicense.com/licenses/.

[19] Frequently Asked Questions about the GNU Licenses, Free Software Foundation, Inc. Saatavissa (viitattu 04.07.2018): https://www.gnu.org/licenses/gpl-faq.en.html#LGPLStaticVsDynamic.

[20] H. Cummings, Learning Node.js for .NET Developers, Packt Publishing, Birmingham, Yhdistynyt kuningaskunta, 2016, 248 p.

[21] Guides, Node.js Foundation. Saatavissa (viitattu 01.05.2018): https://nodejs.org/en/docs/guides/.

[22] NPM documentation, npm. Saatavissa (viitattu 01.05.2018): https://docs.npmjs.com/all.

[23] D. Herron, Node.js Web Development - Third Edition, Packt Publishing, Birmingham, Yhdistynyt kuningaskunta, 2016, 282 p.

[24] E. Rossignon NodeOPCUA, GitHub. Saatavissa (viitattu 01.05.2018): http://node-opcua.github.io/.

[25] M. Karaila Node-RED OPC UA, GitHub. Saatavissa (viitattu 01.05.2018): https://github.com/mikakaraila/node-red-contrib-opcua.

[26] Express - Node.js web application framework, Node.js Foundation. Saatavissa (viitattu 01.05.2018): https://expressjs.com/.

[27] H. Yaapa, Express Web Application Development, 1st ed. Packt Publishing, Olton, 2013, 293 p.

[28] Sequelize: The node.js ORM for PostgreSQL, MySQL, SQLite and MSSQL, Sequelize. Saatavissa (viitattu 01.05.2018): http://docs.sequelizejs.com/.

[29] V. Wang, F. Salim, P. Moskovits, M. Jabali, I. Books24x7, The Definitive Guide to HTML5 WebSocket, 1st ed. Apress, Berkeley, CA, 2014, 208 p.

[30] Socket.IO, Socket.IO. Saatavissa (viitattu 01.05.2018): https://socket.io/.

[31] jQuery, The jQuery Foundation. Saatavissa (viitattu 01.05.2018): http://jquery.com/.

[32] B. Bibeault, Y. Katz, jQuery in action, 3rd ed. Manning, Greenwich, CT, 2015, 504 p.

[33] M. Wendt Fancytree, GitHub. Saatavissa (viitattu 01.05.2018): https://github.com/mar10/fancytree.

[34] winston, C. Robbins. Saatavissa (viitattu 01.05.2018): https://github.com/winstonjs/winston.